Towards Ubiquitous Serverless Computing: Fast Large-Scale Machine Learning and Optimal Pricing for the Cloud

Vipul Gupta

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-80
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-80.html

May 14, 2021

Towards Ubiquitous Serverless Computing: Fast Large-Scale Machine Learning and Optimal Pricing for the Cloud

by

Vipul Gupta

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Kannan Ramchandran, Co-chair
Professor Thomas Courtade, Co-chair
Professor Michael W. Mahoney
Professor Joseph E. Gonzalez

Spring 2021

Towards Ubiquitous Serverless Computing: Fast Large-Scale Machine Learning and Optimal Pricing for the Cloud

Abstract

Towards Ubiquitous Serverless Computing: Fast Large-Scale Machine Learning and Optimal Pricing for the Cloud

by

Vipul Gupta

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Kannan Ramchandran, Co-chair

Professor Thomas Courtade, Co-chair

Serverless computing platforms represent the fastest-growing segment of cloud services and are predicted to dominate the future of cloud computing. However, the real-world applications of serverless systems are somewhat constrained due to several inherent bottlenecks such as their stateless nature, frequent occurrence of stragglers, naive resource allocation and pricing of computing resources, etc. The broader aim of this dissertation is to propose techniques to mitigate such bottlenecks and make the use of serverless computing ubiquitous. In particular, we focus on four applications that relate to large-scale serverless computing and describe them as follows.

Total time for end-to-end distributed computation in such systems suffers due to a subset of slower workers, also referred to as stragglers. First, we propose straggler mitigation schemes for large-scale numerical linear algebra on serverless systems. Serverless systems allow users to massively scale the number of workers, but the total computation time is further exacerbated by high-communication latencies and the stateless nature of Function-as-a-Service (FaaS) platforms. Second, we further propose algorithms for large-scale convex optimization that are amenable to serverless systems. Recently, serverless computing — due to its elasticity, scale, and ease of management — has garnered significant traction from the industry and academia as a platform to train deep neural networks. Third, we further propose algorithms for massive scale non-convex optimization for training deep neural networks that take advantage of the scale of the serverless platform while mitigating communication costs. Finally, any ubiquitous platform is not viable until it serves the needs of the masses. In game-theoretic terms, a commercial platform should provide the required quality of services to customers while maximizing their utility (or satisfaction). Fourth, we propose principled yet practical schemes for allocating and pricing resources in serverless platforms.

To my parents, Saroj and Ram Gopal Gupta.

Contents

Contents **ii**

List of Figures **v**

List of Tables **x**

1 Introduction **1**
1.1 Serverless Computing: Promises and Pitfalls 2
1.2 Numerical Linear Algebra 4
1.3 Convex Optimization 6
1.4 Non-convex Optimization 8
1.5 Resource Allocation and Pricing 9

I Speeding up Linear Algebra **11**

2 Codes for Serverless Straggler Mitigation **12**
2.1 Introduction 12
2.2 Straggler Resilience in Serverless Computing Using Codes 13
2.3 Theoretical Analysis of Local Product Codes 17
2.4 Coded Computing in Applications 22
2.5 Proofs 27
2.6 Conclusions and Future Work 29

3 Sketch-based Serverless Straggler Mitigation **30**
3.1 Introduction 30
3.2 Preliminaries 32
3.3 Cost Analysis: Naive and Blocked multiplication 33
3.4 OverSketch: Straggler-resilient Blocked Matrix Multiplication using Sketching . . 37
3.5 Experimental Results 43
3.6 Proofs 45
3.7 Conclusion 49

II Speeding up Convex Optimization **50**

4 OverSketched Newton **51**
4.1 Introduction . . . 51
4.2 Newton's Method: An Overview . . . 52
4.3 OverSketched Newton . . . 53
4.4 OverSketched Newton on Serverless Systems: Examples . . . 60
4.5 Experimental Results . . . 64
4.6 Proofs . . . 71
4.7 Conclusions . . . 80

5 LocalNewton **81**
5.1 Introduction . . . 81
5.2 Problem Formulation . . . 84
5.3 Algorithms . . . 85
5.4 Convergence Guarantees . . . 89
5.5 Empirical Evaluation . . . 92
5.6 Proofs . . . 95
5.7 Conclusion . . . 106

6 BEAR: Feature Selection in Sublinear Memory **107**
6.1 Introduction . . . 107
6.2 Review: Count Sketch . . . 109
6.3 Stochastic Sketching for Feature Selection . . . 109
6.4 Challenges of Second-order Sketching in Ultra-High Dimension . . . 111
6.5 The BEAR Algorithm . . . 111
6.6 Simulations . . . 114
6.7 Experiments . . . 115
6.8 Proofs . . . 119
6.9 Discussion and Conclusion . . . 123

III Speeding up Non-Convex Optimization **124**

7 Stochastic Weight Averaging in Parallel **125**
7.1 Introduction . . . 125
7.2 Related Work . . . 126
7.3 Stochastic weight averaging in parallel . . . 127
7.4 Loss Landscape Visualization around SWAP iterates . . . 127
7.5 Experiments . . . 130
7.6 Conclusions and Future Work . . . 135

8 Dynamic Communication Thresholding **137**

8.1 Introduction . . . 137
8.2 Communication-Efficient Training with Hybrid Parallelism . . . 140
8.3 Empirical Results . . . 146
8.4 Compression Schemes That Do Not Work . . . 153
8.5 Conclusions . . . 154

IV Resource Allocation and Pricing 155

9 Expected Utility Theory Perspective 156
9.1 Introduction . . . 156
9.2 Problem Formulation . . . 160
9.3 Analyzing SYS-LP . . . 165
9.4 Pricing Mechanisms to Learn User Utilities . . . 167
9.5 A Market Simulation . . . 170
9.6 Future work . . . 174
9.7 Related Work . . . 175
9.8 Proofs . . . 176
9.9 Remarks . . . 180

10 Cumulative Prospect Theory Perspective 182
10.1 Introduction . . . 182
10.2 Model . . . 185
10.3 Analysis . . . 191
10.4 Pricing Mechanisms for Lottery-based Allocation . . . 198
10.5 Experiments . . . 202
10.6 Proofs . . . 207

Bibliography 212

List of Figures

1.1 Example of a serverless job where the user uploads a function and data and the cloud launches worker(s) that map the function onto the dataset while abstracting away the details of the server management from the user. 2
1.2 Job times of AWS Lambda workers for distributed matrix multiplication. 4

2.1 Typical workflow on a serverless system for computing the matrix multiplication $\mathbf{AB}^T$. Here, f_{enc}, f_{comp} and f_{dec} denote the functions corresponding to encoding, computation, and decoding, respectively, that are employed at the serverless workers (in parallel on different data points). Whereas most existing schemes focus on minimizing time required to compute the product (T_{comp}), our focus is on minimizing the end-to-end latency that involves parallel encoding (T_{enc}) and decoding (T_{dec}) times as well. 13
2.2 Coded computing versus speculative execution for power iteration on a matrix of dimension 0.5 million for 20 iterations. 14
2.3 Computing $\mathbf{C} = \mathbf{AA}^T$ where $\mathbf{A}$ is divided into four row-blocks and $L_A = 2$. Here, $\mathbf{C}_{ij} = \mathbf{A}_i\mathbf{A}_j^T$. Locally encoding the rows of $\mathbf{A}$ leads to a locally recoverable code in the output $\mathbf{C}_{\text{coded}}$. 15
2.4 Comparison of average runtimes of proposed schemes versus existing schemes for multiplication of two square matrices. For large matrix dimensions, decoding with a polynomial code is not feasible since the master node cannot store all the data locally. 16
2.5 Probabilistic upper bound on the number of blocks read, R, by a decoding worker from Theorem 1 shown for $L = 10$, $n = 121$, and $p = 0.02$. Here, $\Pr(R \geq 2\mathbb{E}[R]) \leq 3.1 \times 10^{-3}$. 20
2.6 Some examples of undecodable sets, as viewed from a single decoding worker's $(L_A + 1) \times (L_B + 1)$ grid. The yellow blocks correspond to the systematic part of the code, and blue blocks to the parity. Blocks marked with an "X" are stragglers. 20
2.7 Some examples of "interlocking" three straggler configurations. Stragglers can be decoded using a peeling decoder. 21
2.8 Upper bound on probability of the event $\bar{D}$ (that is, a decoding worker being unable to decode) when $p = .02$. We chose $n = 121$ in our experiments which represents a good trade-off between code redundancy and straggler resiliency. 22
2.9 Coded computing versus speculative execution for KRR with PCG on the ADULT dataset. Error on testing dataset was 11%. 23
2.10 Coded computing versus speculative execution for KRR with PCG on the EPSILON dataset. Error on testing dataset was 8%. 23

2.11 Comparison of proposed coding scheme, that is, local product codes, versus speculative execution for straggler mitigation on AWS Lambda. 26

3.1 Matrix $\mathbf{A}$ is divided into 2 row chunks $\mathbf{A}_1$ and $\mathbf{A}_2$, while $\mathbf{B}$ is divided into two column chunks $\mathbf{B}_1$ and $\mathbf{B}_2$. During the encoding process, redundant chunks $\mathbf{A}_1 + \mathbf{A}_2$ and $\mathbf{B}_1 + \mathbf{B}_2$ are created. To compute $\mathbf{C}$, 9 workers store each possible combination of a chunk of $\mathbf{A}$ and $\mathbf{B}$ and multiply them. During the decoding phase, the master can recover the affected data (C_{11}, C_{12} and C_{22} in this case) using the redundant chunks. 32
3.2 An illustration of two algorithms for distributed matrix multiplication. 33
3.3 Comparison of AWS Lambda costs for multiplying two $n \times n$ matrices, where each worker is limited by 3008 MB of memory and price per running worker per 100 milliseconds is $\$0.000004897$. 36
3.4 An illustration of multiplication of $m \times z$ matrix $\tilde{\mathbf{A}}$ and $z \times l$ matrix $\tilde{\mathbf{B}}$, where $z = d + b$ assures resiliency against one straggler per block of $\mathbf{C}$, and d is chosen by the user to guarantee a desired accuracy. Here, $m = l = 2b$, $d = 3b$, where b is the block-size for blocked matrix multiplication. This scheme ensures one worker can be ignored while calculating each block of $\mathbf{C}$. 38
3.5 An illustration of sketching $\mathbf{A} \in \mathbb{R}^{m \times n}$ in parallel using the sketch matrix in Eq. (3.3) with sketch dimension $z = (N + e)b$. Worker W_{ij} receives the row-block $\mathbf{A}(i, :)$ of $\mathbf{A}$ and the Count-sketch $\mathbf{S}_j$ to compute the (i, j)-th block of $\tilde{\mathbf{A}}$. Sketching requires a total of mz/b^2 workers. Here, $z = 4b, N = 3$ and $e = 1$, and $\mathbf{A}$ is divided into 2 row-blocks, that is, $m = 2b$. Total number of workers required for distributed sketching is 8. 38
3.6 Time and approximation error for OverSketch with 3000 workers when e, the number of workers ignored per block of $\mathbf{C}$, is varied from 0 to 10. 42
3.7 Time statistics and optimality gap on AWS Lambda while solving the LP in (3.5) using interior point methods, where e is the number of workers ignored per block of $\mathbf{C}$. 42
3.8 Comparison of OverSketch with coded theory based scheme in [26] on AWS Lambda. OverSketch requires asymptotically less workers which translates to significant savings in cost. . . . 45

4.1 **Coded matrix-vector multiplication**: Matrix $\mathbf{A}$ is divided into 2 row chunks $\mathbf{A}_1$ and $\mathbf{A}_2$. During encoding, redundant chunk $\mathbf{A}_1 + \mathbf{A}_2$ is created. Three workers obtain $\mathbf{A}_1$,$\mathbf{A}_2$ and $\mathbf{A}_1 + \mathbf{A}_2$ from the cloud storage S3, respectively, and then multiply by $\mathbf{x}$ and write back the result to the cloud. The master M can decode $\mathbf{Ax}$ from the results of any two workers, thus being resilient to one straggler (W_2 in this case). 54
4.2 **OverSketch-based approximate Hessian computation:** First, the matrix $\mathbf{A}$—satisfying $\mathbf{A}^T\mathbf{A} = \nabla^2 f(\mathbf{w}_t)$—is sketched in parallel using the sketch in (4.4). Then, each worker receives a block of each of the sketched matrices $\mathbf{A}^T\mathbf{S}$ and $\mathbf{S}^T\mathbf{A}$, multiplies them, and communicates back its results for reduction. During reduction, stragglers can be ignored by the virtue of "over" sketching. For example, here the desired sketch dimension m is increased by block-size b for obtaining resiliency against one straggler for each block of $\hat{\mathbf{H}}$. 54

4.3 GIANT: The two stage second order distributed optimization scheme with four workers. First, master calculates the full gradient by aggregating local gradients from workers. Second, the master calculates approximate Hessian using local second-order updates from workers. 65
4.4 Different gradient descent schemes in serverful systems in presence of stragglers . . . 65
4.5 Convergence comparison of GIANT (employed with different straggler mitigation methods), exact Newton's method and OverSketched Newton for Logistic regression on AWS Lambda. The synthetic dataset considered has 300,000 examples and 3000 features. 66
4.6 Comparison of training and testing errors for logistic regression on EPSILON dataset with several Newton based schemes on AWS Lambda. OverSketched Newton outperforms others by at least 46%. Testing error closely follows training error. 67
4.7 Logistic regression on WEBPAGE and a9a datasets with several Newton based schemes on AWS Lambda. OverSketched Newton outperforms others by at least 25%. 68
4.8 Convergence comparison of gradient descent, exact Newton's method and OverSketched Newton for Softmax regression on AWS Lambda. 69
4.9 Convergence comparison of speculative execution and coded computing for gradient and Hessian computing with logistic regression on AWS Lambda. 69
4.10 Convergence comparison of gradient descent, NAG and OverSketched Newton on AWS Lambda. 69
4.11 Convergence comparison of GIANT on AWS EC2 and OverSketched Newton on AWS Lambda. 69

5.1 Training loss and testing accuracy versus communication rounds for Adaptive LocalNewton and existing schemes for communication-efficient optimization. 82
5.2 Comparing LocalNewton (for different values of L) and GIANT. In general, LocalNewton reaches very close to the optimal solution. In that region, however, the convergence rate of LocalNewton is slow. This is mitigated by using Adaptive LocalNewton which appends the LocalNewton iterations with better quality (but more expensive) updates from GIANT. 88
5.3 Experiments on the different datasets from Table 5.1 on AWS Lambda. Both in terms of training loss and testing accuracy, Adaptive LocalNewton converges to the optimal value at least 50% faster than existing schemes. 94
5.4 Training times (red bars) and communication rounds (blue bars) required to reach the same training loss of $0.19, 0.65$ and 0.3 for the w8a, Covtype and EPSILON datasets, respectively, on AWS Lambda. Here, A: Adaptive LocalNewton, B: BFGS, C: Local SGD, D: GIANT. 95

6.1 Feature selection experiments on $p = 1000$-dimensional synthetic data sets with entries drawn from normal distribution. A) Probability of success in recovering all features correctly as a function of compression factor. B) Recovery error rate in terms of the ℓ_2-norm. C) Probability of success as a function of the value of the step size (compression factor $= 2.22$). 115

6.2 Classification performance as a function of the compression factor in real-world data sets. 117
6.3 Classification performance as a function of the number of the top-k features. 117

7.1 Learning rate schedules and CIFAR10 test accuracies for workers participating in SWAP. The large-batch phase with synchronized models is followed by the small-batch phase with diverging independent models. The test accuracy of the averaged weight model is computed by averaging the independent models and computing the test loss for the resulting model. 128
7.2 CIFAR10 train and test error restricted to a 2D plane spanned by the output of phase 1 ('LB') one of the outputs of phase 2 ('SGD') and the averaged model ('SWAP'). 130
7.3 CIFAR10 train and test error restricted to a 2D plane spanned by the output of three workers after phase 2 ('SGD1', 'SGD2', 'SGD3') and location of the average model ('SWAP'). The minimum test error achievable for models restricted to this region of the plane (marked as *BEST*). 130
7.4 Cosine similarity between direction of gradient descent and $\Delta\theta$ 131
7.5 Learning rate and mini-batch schedules used for ImageNet. The original schedule for 8 GPUs was taken from an existing DAWNBench submission. For a larger batch experiment, we double the batch size, double the number of GPUs and double the learning rate of the original schedule. For SWAP, we switch from the modified schedule to the original schedule as we move from phase 1 to phase 2. 133
7.6 Illustration of SWA with different batch sizes 134

8.1 Distributed DNN training with hybrid training which uses both DP (left) and MP (right) for greater parallelization gains. During DP, multiple trainers process several mini-batches of data in parallel. During MP, one copy of the model is processed by one trainer which in turn is comprised of multiple workers. 138
8.2 Top-K threshold for various levels of sparsity during the gradient compression for DCT-DP. We see that the top-K thresholds, for different sparsity levels, do not deviate much from the mean. Thus, updating the threshold only every $L(> 1)$ iterations can help reduce the overhead of sorting to find the top-K threshold. 141
8.3 A illustration of DCT-DP. First, $W_{grad} \in \mathbb{R}^N$ (which already incorporates error from the previous iteration) is compressed using a threshold τ to obtain the sparse vector $\widehat{W}_{grad}$. Then, the error is calculated as $E_{grad} = W_{grad} - \widehat{W}_{grad}$ to be used in the next iteration to correct the error in gradient direction. 141
8.4 A illustration of DCT-MP. During the forward pass, we sparsify and compress the activations, say X_{act}, corresponding to one data sample, using the mask, $\mathbb{I}(|X_{act}| \geq \tau)$, is generated based on the threshold τ. During the backward pass, the same mask is used to compress the gradients and selectively train neurons. 142
8.5 Top-K threshold for various levels of sparsity for the cases when compression using DCT-MP is applied and when it is not applied. The top-K thresholds decrease significantly when DCT-MP is applied. Thus, DCT-MP induces sparsity in neuron activations. This is possibly the reason for its improved generalization performance. 144

8.6 Using Laplace distribution to model the entries of activation matrix and using it to predict the threshold values. . . . 144
8.7 A illustration of model parallelism with DLRM. The entities that are sent across the network are shown in red. X_{act} and X_{grad} are communicated during MP, and W_{grad} is communicated during DP. The shaded area in blue represents a sample model partitioning for MP. In this case, three workers are working on one copy of the model during MP and comprise a trainer. . . . 147
8.8 DCT-DP on Large-Scale Recommendation Models. Figures (a) and (b) show the training time and loss improvements, respectively, over baseline for different values of the threshold life-span, L, for a sparsity level of 95%. Figures (c) and (d) show the same statistics for various levels of sparsity for $L = 1000$. . . . 151

9.1 Examples of utility functions for 3 users that depend on completion time. . . . 158
9.2 A block diagram schematic of the system with N users and T service tiers 158
9.3 Percentage error between V^R (the utility obtained with SYS-LP problem) and $\hat{V}$ (the utility obtained by applying integer constraints on SYS-LP solution). . . . 166
9.4 Decomposition into user and cloud problems. Here, we visualize a user black-box for each user i that solves the $\mathrm{USER}(i)$ problem using the prices shown by the cloud black-box, the unscheduled functions in the job of user i and her willingness to pay (intermittently updated). The cloud black-box runs Algorithm 19 to update the prices using the budget signals from the user black-boxes and the capacity constraints from the cloud. . . . 171
9.5 Finding the SYS-LP solution through price tracking (Algorithm 19). The algorithm converges after $\sim$10 iterations. . . . 172
9.6 Sum utilities for three resource allocation schemes 172
9.7 Utility and price charged in the first tier 173
9.8 Utility and price paid by a randomly chosen user 173

10.1 A block diagram schematic for the system problem 183
10.2 Examples of probability weighting functions. Here, we have used the form $w_i(p) = \frac{\delta p^\gamma}{\delta p^\gamma + (1-p)^\gamma}$, suggested by [269] with $\delta = 0.7$ and γ as indicated in the plot. . . . 186
10.3 An illustration of how $h_i(\cdot)$ is calculated using the CPT weighing function $w_i(\cdot)$. . . . 192
10.4 Performance of the tracking-based algorithm. . . . 202
10.5 CPT versus EU Utilities when the number of users (that is, demand) is increased. . . . 204
10.6 CPT versus EU Utilities when the percentage of users that exhibit CPT behavior is increased. . . . 204
10.7 Sum utilities for the four resource allocation schemes 205
10.8 Resource allocation statistics for a low-utility user 206
10.9 Resource allocation statistics for a high-utility user 206
10.10 Price tracking using only one user-budget update per day is close to optimal for the CPT case. Further, the average revenue of the Cloud increased with the CPT allocation while maximizing the system utility. . . . 207

List of Tables

3.1 Costs comparison for naive and blocked matrix multiplication in the serverless setting, where $\delta < 2$. 36

5.1 Datasets considered for experiments in this chapter 93
5.2 Step-sizes obtained using tuning for Local SGD and BFGS for several datasets 93

6.1 Memory cost of the vectors in BEAR. 113
6.2 Summary of the real-world data sets. 116
6.3 Examples of the features selected in RCV1. 119
6.4 Overall run time comparison (minutes). 119

7.1 Training Statistics for CIFAR10 131
7.2 Training Statistics for CIFAR100 132
7.3 Training Statistics for ImageNet 133
7.4 Comparison: SWA versus SWAP 134
7.5 Hyperparameters obtained using tuning for CIFAR10 135
7.6 Hyperparameters obtained using tuning for CIFAR100 135

8.1 DCT-MP on the DLRM model with the Criteo Ad Kaggle dataset: Finding the top-K threshold using Laplacian estimation. 145
8.2 DCT-MP on the DLRM model: Train and Test Loss and Accuracy for multiple sparsity ratios (denoted by η) and different settings for MP. 148
8.3 DCT-DP on the DLRM model: Train and Test Loss and Accuracy for various levels of sparsity. 148
8.4 Compression using DCT-DP and DCT-MP on the DLRM model: Train and Test Loss and Accuracy with two MP splits (that is, three workers for MP). 149
8.5 DCT-MP on a translation model with IWSLT'14 dataset: Train and Test Losses and BLEU scores for various levels of sparsity and different splits for MP. 150
8.6 DCT-MP on a large-scale recommender model 152
8.7 Compressing the activation matrix during MP using Gaussian sketching does not yield good results. 153
8.8 Compressing the gradient matrix during backward pass in MP using top-K sparsification does not yield good results. 154

Acknowledgments

The five years at Berkeley went by very quickly. This place has taught me so much, both technically and otherwise, for which I will always be grateful. I will try my best to express gratitude to people who have helped me during this journey.

A lot of this learning is due to my advisors Prof. Kannan Ramchandran and Prof. Thomas Courtade. Both professionally and personally, I have learnt a great deal from Kannan, be it from the polite manner in his emails, or his cheerful body language during the meetings, or his enthusiasm to always learn new things and dig in further to make an impact. Kannan was the one who invited me to Berkeley and motivated me to start collaborating from the very first day. He has always been available for brainstorming sessions, and if I have a crazy idea, he would be the first one I would talk to without any reservations. I feel that most of my research problems are motivated by Kannan's thinking and I think he is going to motivate many more problems I will work on in the future. I am particularly inspired by his characteristic of keeping an eye out for the "bigger picture", a quality that I think will serve me well in my career.

Tom has been a great source of motivation and support in my entire Ph.D. life. I could generally speak up my mind freely around him and tell him anything that bothers me, and I would always receive kind words of motivation in return. His problem-solving skills are mind-boggling. I can go to him with any unsolved problem and would come out of the meeting with concrete directions to proceed, if not solving the entire problem. He has been kind enough to allow me to work on problems that interest me but are unrelated to his primary area of research, and in fact, approach these new problems with the same enthusiasm as me. Tom also played a pivotal role in the writing and presentation of some of my early papers.

I would also like to thank my "pseudo" co-advisor and thesis committee member Prof. Michael Mahoney. I will always be grateful for him treating me as his student, more so since he was not required to do that in any formal capacity. Michael kindly invited me to his group meetings and has always been available to discuss new problems to work on and collaborate on. Michael, like Kannan, is always ready to dive into new research areas, explore new problems, and form new collaborations to get to the end of any research problem. Throughout the Ph.D., I have collaborated with multiple members of his amazing group. Also, Michael has taught me how to think about the long-term goals when starting to work on a project–what would be its impact, what audience does it cater to, and what is the best way to write and present the results. In this age of redundant submissions and noisy reviews, it's extremely important to flush out these questions to be successful.

Among other professors whom I had a chance to interact with at Berkeley, I will mention a few more–Prof. Joseph Gonzalez and Prof. Aditya Guntuboyina for being members of my Qual committee and providing feedback halfway through the thesis; and Prof. Venkat Anantharam, Prof. Anant Sahai, and Prof. Martin Wainwright for being amazing teachers and a great presence around the BLISS lab.

One defining quality of Berkeley is the extensive scale of collaborations. I am extremely grateful to my primary research collaborators—Swanand Kadhe, Yaoqing Yang, Soham Phade, Dominic Carrano, Avishek Ghosh, and Amirali Aghazadeh—for making research fun and interesting. The

above list of collaborators is, by no means, exhaustive. Working with a diverse set of talented people was a great learning experience in itself.

It was great to have a workplace surrounded by super helpful, relentlessly fun, and amazingly genius folks at BLISS lab. For this *blissful* time, thanks to Ashwin Pananjady, Avishek Ghosh, Nihar Shah, Rashmi Vinayak, Swanand Kadhe, Soham Phade, Aviva Bulow, Sang-min Han, Fanny Yang, Vidya Muthukumar, Payam Delgosha, Jingge Zhu, Amirali Aghazadeh, Raaz Dwivedi, Efe Aras, Kuan-Yun Lee, Vignesh Subramanian, Banghua Zhu, and Nived Rajaraman. Special thanks to Ashwin for being my peer advisor and helping me navigate the Berkeley system during the first few months of my Ph.D. I should also mention the very cool undergraduates that worked with us: Kabir Chandrashekhar, Tavor Baharav, Mitas Ray, Nikunj Jain, and Dominic Carrano. The pandemic made me realize how much I would miss hanging out with the BLISS lab.

I would also like to acknowledge the efforts of the EECS administrative staff in resolving many administrative issues, and particularly, Shirley Salanio who is always ready to walk the extra mile to help students out.

Five years at Berkeley were some of the most amazing memories, and that would not have been possible if not for many friends outside of work. Ultimate frisbees on Fridays gave us another reason to look forward to the end of the week. Prominent frisbee enthusiasts (apart from the BLISS folks) include Bala TK, Prasanth Kotaru, Karan Jain, Nathan Bucki, Mike Danielczuk, and Jonathan Lee. I am extremely grateful to my past and current housemates Avishek Ghosh, Aviva Bulow, Mohit Bhura, Gurudayal, Niharika Gupta, and Harsh Gupta for tolerating me and feeding me all these years. Special thanks to Aviva to motivate me to participate and complete the Lake Tahoe triathlon and for being the partner-in-crime in all those training sessions. It was undoubtedly the most physically strenuous adventure I have undertaken. Also, the Indian graduate students group never made me feel far from home, and for this and several other cooking and traveling adventures, I am thankful to Shruti Bhatia, Anamika Chowdhury, Milind Hegde, Bala TK, Prasanth Kotaru, and Koulik Khamaru. Koulik deserves a special mention for being the to-go guy to travel to any place inside or outside the continent. I would also like to mention my friends from IITK, Harsh Gupta and Dhruv Kumar Yadav, for making life easier at the start of the pandemic, and later, learning to play tennis with pros such as Yeshwanth and Devendra gave the much-needed respite.

And last, but not least, I am ever grateful to my family, and particularly my parents, Saroj and Ram Gopal Gupta, for their selfless love and unwavering trust in my abilities. I owe all my achievements to their unconditional support and motivation for me to pursue my dreams. This thesis is dedicated to them.

Chapter 1

Introduction

In recent years, there has been tremendous growth in users performing distributed computing operations on the cloud, largely due to extensive and inexpensive commercial offerings like Amazon Web Services (AWS), Google Cloud, Microsoft Azure, etc. We focus on a recently introduced cloud service called *serverless computing* for general distributed computation. Traditional "serverful" computing allows customers to rent servers from the cloud for a fixed duration, and the customers can access the machines (say, through an IP address) and use it like a personal computer.

Serverless platforms differ from serverful systems by providing programming abstractions, thus, simplifying the process of writing software for the cloud. Also called "Function-as-a-Service (FaaS)", the idea is that the user submits functions to the cloud which when triggered are evaluated by serverless workers, which in turn are managed by the cloud provider. An example is illustrated in Fig. 1.1, where the user is uploading job which consists of a function (e.g., matrix-matrix multiplication) and data to the cloud storage. The job when triggered launches serverless workers that map the function onto the dataset and return the output to the cloud storage. The user can then ping the cloud to get back the results of the computation.

Serverless systems have garnered significant attention from industry (e.g., Amazon Web Services (AWS) Lambda, Microsoft Azure Functions, Google Cloud Functions) as well as the research community (see, e.g., [1]–[9]). Serverless platforms[1] penetrate a large user base by removing the need for complicated cluster management while allocating resources expeditiously which provides greater elasticity and easy scalability [1]–[3]. For these reasons, serverless systems are expected to abstract away today's cloud servers in the coming decade just as cloud servers abstracted away physical servers in the past decade [7]–[10].

Due to its massive scalability and convenience in operation, the use of serverless systems is gaining significant research traction. It is forecasted that the market share of serverless will grow by USD 9.16 billion during 2019-2023 (at a CAGR of 11%) [11]. Indeed, according to the *Berkeley view on Serverless Computing* [7], serverless systems are expected to dominate the cloud scenario and become the default computing paradigm in the coming years while client-server based computing will witness a considerable decline. For these reasons, using serverless systems for

[1]The name serverless is an oxymoron since all the computing is still done on servers, but the name stuck as it abstracts away the need to provision or manage servers.

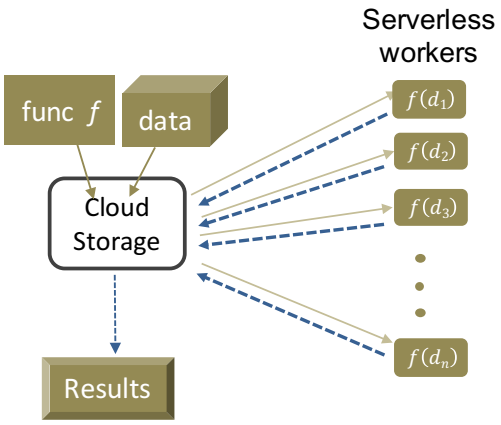


Figure 1.1: Example of a serverless job where the user uploads a function and data and the cloud launches worker(s) that map the function onto the dataset while abstracting away the details of the server management from the user.

large-scale computation has garnered significant attention from the systems community [1], [3], [9], [12]–[16].

1.1 Serverless Computing: Promises and Pitfalls

In this section, we outline some distinguishing properties of serverless systems. First, we describe some advantages of serverless platforms that make it enticing for users.

1. **Simple abstraction for user**: Although servers are still the ones powering all the computation, serverless computing manages to hide these servers from the programmers and create a simple abstraction that allow them to write code efficiently.

2. **Massive Scale and Autoscaling**: Unlike *serverful* computing, the number of inexpensive workers in serverless platforms is flexible and scales automatically, often scaling into the thousands [1], [9]. This allows users to execute algorithms that demand dynamic parallelism without wasting compute resources, for example, distributed Cholesky decomposition [17]. The scalability of the serverless systems is further exemplified by the fact that we can launch more than 16,000 machines within 10 seconds on AWS Lambda [3].

3. **Pay-as-you-go cost model**: Serverless platforms offer pay-as-you-go cost model instead of a reservation-based model. Thus, there's no cost for idle resources. This, combined with the autoscaling nature makes serverless enticing if the customers need the machines on-demand but only for a shorter duration.

The serverless platform enjoys the aforementioned features at the cost of several bottlenecks, described in detail below.

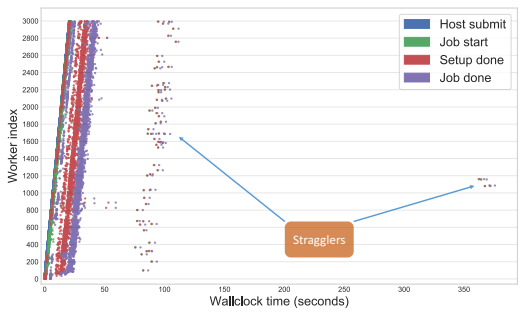
1. **High Communication Latency**: Serverless frameworks suffer from high communication latency between worker machines dominates the running time of the algorithm in general [4], [6]. These systems use cloud storage (e.g., AWS S3) to store enormous amounts of data, while using a large number of low-quality workers for large-scale computation. Naturally, the communication between the high-latency storage and the commodity workers is extremely slow [e.g., see 1], resulting in impractical end-to-end times for many popular optimization algorithms such as SGD [4], [6].

2. **Stragglers and Faults**: Unlike HPC/serverful systems, nodes in the serverless systems suffer degradation due to what is known as *system noise*. This can be a result of limited availability of shared resources, hardware failure, network latency, etc. [18], [19]. This results in job time variability, and hence a subset of much slower nodes, often called *stragglers*. These stragglers significantly slow the overall computation time, especially in large or iterative jobs. In Fig. 1.2, we plot the running times for a distributed matrix multiplication job with 3600 workers on AWS Lambda and demonstrate the effect of stragglers on the total job time. In fact, our experiments consistently demonstrate that at least 2% workers take significantly longer than the median job time, severely degrading the overall efficiency of the system. The effects are exacerbated for many common iterative distributed algorithms (such as gradient descent, conjugate gradient, etc.) due to accumulation of tail times for stragglers in each iteration.

3. **Ephemeral and Low-memory workers**: Unlike serverful computing, the number of inexpensive workers can quickly scale into the thousands [1]. This heavy gain in the computation power, however, comes with the disadvantage that the commodity workers in serverless architecture are ephemeral and have low memory.[2] Thus, if we are running a distributed iterative algorithm, the ephemeral nature of the serverless workers requires that new workers should be invoked every few iterations and data should be communicated to them.

4. **Resource Allocation and Pricing**: Serverless computing is a new ecosystem with a growing customer base. Currently, all the jobs are allocated on a uniform basis by the cloud provider without taking job requirements into account. For example, *urgent* jobs (that need to be executed in real-time, such as model deployment [12] and real-time video compression [20]) may need prioritization, whereas *enduring* jobs (that can be put into queues with reasonable wait-times, such as optimization in machine learning [6], [14], [16], [21] and scientific computing [3], [5], [8]) could be put on hold. Further, since serverless platform offers pay-as-you-go cost model, proper resource allocation and incentive-based pricing are even more important since any idle resources cost money and may eventually result in decreased profit margins for the cloud provider. Hence it is extremely desirable, and perhaps overdue, that we have a resource allocation scheme that schedules and prices resources while taking customer requirements into account. Further, the current pricing scheme is constant and independent of

[2]For example, serverless nodes in AWS Lambda have a maximum memory of 10 GB (up from 3 GB) and a maximum runtime of 900 seconds (up from 300 seconds). Note that these numbers may change as the serverless platform matures.



(a) Job times for 3000 AWS Lambda nodes in one trial. The median job time is 40 seconds, and $\sim$ 5% of the nodes take 100 seconds, and two nodes take as much as 375 seconds for the same job.

(b) Average job times for 3600 AWS Lambda nodes over 10 trials for distributed matrix multiplication. The median job time is around 135 seconds, and around 2% of the nodes take up to 180 seconds on average.

Figure 1.2: Job times of AWS Lambda workers for distributed matrix multiplication.

the market. Ideally, the pricing scheme should represent an interplay between the demand and supply and adapt prices to the demand of the cloud resources.

In this dissertation, we propose techniques for large-scale scientific computing and machine learning that mitigate serverless bottlenecks such as stragglers, high-communication latency, low-quality of workers while utilizing their massive scale and flexibility. Additionally, we develop resource allocation and pricing schemes for the serverless platform from an economist's perspective that take user feedback into account while scheduling a job on the cloud. In particular, we divide the dissertation into the following four parts, each of which focuses on an important application: (1) Numerical Linear Algebra, (2) Large-scale convex optimization, (3) Large-scale non-convex optimization, and (4) Resource allocation and pricing for the serverless platform. We describe these parts in detail next.

1.2 Numerical Linear Algebra

Part I contains two chapters which focus on large-scale serverless computing linear algebra applications. Our techniques capitalize on the scale and elasticity of serverless computing while mitigating bottlenecks such as the occurrence of stragglers.

Techniques like speculative execution have been traditionally used to deal with stragglers (e.g., in Hadoop MapReduce [22] and Apache Spark [23]). Speculative execution works by detecting workers that are running slowly, or will slow down in the future, and then assigning their jobs to new workers without shutting down the original job. The worker that finishes first submits its results. This has several drawbacks: constant monitoring of jobs is required, which is costly when the number of workers is large. Monitoring is especially difficult in serverless systems where worker management is done by the cloud provider and the user has no direct supervision over the

workers. Moreover, it is often the case that a worker straggles only at the end of the job (say, while communicating the results). By the time the job is resubmitted, the additional communication and computational overhead would have decreased the overall efficiency of the system. The situation is even worse for smaller jobs, as spinning up an extra node requires additional invocation and setup time which can exceed the original job time.

Error correcting codes are a linchpin of digital transmission and storage technologies, vastly improving their efficiency compared to uncoded systems. Recently, there has been a significant amount research focused on applying coding-theoretic ideas to introduce redundancy into distributed computation for improved straggler and fault resilience, for example, see [24]–[39]. This line of work focuses on cloud computing models consistent with first-generation cloud platforms (i.e., "serverful" platforms), where the user is responsible for node management through a centralized master node that coordinates encoding, decoding and any update phases. Accordingly, most existing schemes typically employ variants of Maximum Distance Separable (MDS) codes, and have focused on optimizing the recovery threshold (i.e., minimum number of machines needed to do a task) of the algorithm, e.g. [28], [29]. This is equivalent to minimizing the compute time while assuming that the encoding/decoding times are negligible. When the system size is relatively small, the encoding/decoding costs can be safely ignored. However, the encoding/decoding costs of such coded computation schemes scale with the size of the system, and hence this assumption does not hold anymore for serverless systems that can invoke tens of thousands of workers [3], [7]. Furthermore, existing schemes require a powerful master with high bandwidth and large memory to communicate and store all the data to perform encoding and decoding locally. This goes against the very idea of massive scale distributed computation. Therefore, coding schemes designed for serverful systems cannot guarantee low end-to-end latency in terms of total execution time for large-scale computation in serverless systems.

In Chapter 2, we propose and implement simple yet principled coding schemes for straggler mitigation in serverless systems for matrix multiplication and evaluate them on several common applications from machine learning and high-performance computing. The proposed coding scheme employs parallel encoding and decoding over the data stored in the cloud using serverless workers. This creates a fully distributed computing framework without using a master node to conduct encoding or decoding, which removes the computation, communication and storage bottleneck at the master. On the theory side, we establish that our proposed scheme is asymptotically optimal in terms of decoding time and provide a lower bound on the number of stragglers it can tolerate with high probability. Through extensive experiments, we show that our scheme outperforms existing schemes such as speculative execution and other coding theoretic methods by at least 25%.

Many of the recent advances in algorithms for numerical linear algebra have come from the technique of linear sketching, in which a given matrix is compressed by multiplying it with a random matrix of appropriate dimension. The resulting product can then act as a proxy for the original matrix in expensive computations such as matrix multiplication, least-squares regression, low-rank approximation, etc. [40]–[43]. For example, computing the product of $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times l}$ takes $O(mnl)$ time. However, if we use $\mathbf{S} \in \mathbb{R}^{n\times d}$ to compute the sketches, say $\tilde{\mathbf{A}} = \mathbf{AS} \in \mathbb{R}^{m\times d}$ and $\tilde{\mathbf{B}} = \mathbf{S}^T\mathbf{B} \in \mathbb{R}^{d\times l}$, where $d \ll n$ is the sketch dimension, we can reduce the computation time to $O(mdl)$ by computing an approximate product $\mathbf{ASS}^T\mathbf{B}$. This is very useful in

applications like machine learning, where the data itself is noisy, and computing the exact result is not needed. For example, in [42], authors propose Newton Sketch, a randomized second order method for optimization that is based on performing an approximate Newton step using Hessian that is calculated by multiplying sketched matrices.

In Chapter 3, we propose OverSketch, an approximate algorithm for distributed matrix multiplication in serverless computing. OverSketch leverages ideas from matrix sketching and high-performance computing to enable cost-efficient multiplication that is resilient to faults and straggling nodes pervasive in low-cost serverless architectures. We establish statistical guarantees on the accuracy of OverSketch and empirically validate our results by solving a large-scale linear program using interior-point methods and demonstrate a 34% reduction in compute time on AWS Lambda.

1.3 Convex Optimization

In part II, we focus on large-scale convex optimization in serverless systems. Our techniques take advantage of the massive scale and autoscaling properties of the serverless platform by mitigating bottlenecks such as limited memory at worker nodes, high communication latency, and frequent occurrence of stragglers. The central theme of these chapters is to utilize the second-order information to reduce the number of iterations to solve the problem, thus overcoming challenges such as higher communication latency and persistent stragglers.

In many machine learning applications, where the data itself is noisy, using the exact Hessian is not necessary. Indeed, using ideas from RandNLA, one can prove convergence guarantees for second-order methods on a single machine, when the Hessian is computed approximately [42], [44]–[46]. To accomplish this, many sketching schemes can be used (sub-Gaussian, Hadamard, random row sampling, sparse Johnson-Lindenstrauss, etc. [41], [47]), but these methods cannot tolerate stragglers, and thus they do not perform well in serverless environments.

There has also been a growing research interest in designing and analyzing distributed implementations of stochastic second-order methods [48]–[53]. However, these implementations are tailored for serverful distributed systems. Our focus, on the other hand, is on serverless systems. Optimization over the serverless framework has garnered significant interest from the research community. However, these works either evaluate and benchmark existing algorithms (e.g., see [13]–[15]) or focus on designing new systems frameworks for faster optimization (e.g., see [16]) on serverless.

In Chapter 4, we develop OverSketched Newton, a randomized Hessian-based optimization algorithm that is tailored to serverless systems to solve for large-scale convex optimization. OverSketched Newton leverages matrix sketching ideas from Randomized Numerical Linear Algebra to compute the Hessian approximately. These sketching methods lead to inbuilt resiliency against stragglers that are a characteristic of serverless architectures. Depending on whether the problem is strongly convex or not, we propose different iteration updates using the approximate Hessian. For both cases, we establish convergence guarantees for OverSketched Newton and empirically validate our results by solving large-scale supervised learning problems on real-world datasets.

Experiments demonstrate a reduction of $\sim 50\%$ in total running time on AWS Lambda, compared to state-of-the-art distributed optimization schemes.

In recent years, schemes such as local SGD have gained popularity, as they are communication-efficient due to only sporadic model updates at the master [54]–[56]. Such schemes that show great promise have eluded a thorough theoretical analysis until recently, when it was shown that local SGD converges at the same rate as mini-batch SGD [57]–[59]. Similar ideas that reduce communication by averaging the local models sporadically have also been applied in training neural networks to improve the training times and/or model performance [60], [61]. Apart from local first-order methods, many communication-efficient distributed second-order (also know as Newton-type) algorithms have been recently proposed [48]–[51], [53], [62]–[65]. Such methods use both the gradient and the curvature information to provide an order of improvement in convergence, compared to vanilla first-order methods. This is done at the cost of more local computation per iteration, which is acceptable for systems with high communication latency. However, such algorithms require at least two communication rounds (for averaging gradients and the second-order descent direction), and a thorough knowledge of a fundamental trade-off between communication and local computation is still lacking for these methods. Stochastic second order optimization theory has been developed recently [44]–[46], and second order implementations motivated by this theory have been shown to outperform state-of-the-art [66], [67].

In Chapter 5, we propose LocalNewton, a distributed second-order algorithm with *local averaging* to address the communication bottleneck problem in distributed optimization. In LocalNewton, the worker machines update their model in every iteration by finding a suitable second-order descent direction using only the data and model stored in their own local memory. We let the workers run multiple such iterations locally and communicate the models to the master node only once every few (say L) iterations. LocalNewton is highly practical since it requires only one hyperparameter, the number L of local iterations. We use novel matrix concentration based techniques to obtain theoretical guarantees for LocalNewton, and we validate them with detailed empirical evaluation. To enhance practicability, we devise an adaptive scheme to choose L, and we show that this reduces the number of local iterations in worker machines between two model synchronizations as the training proceeds, successively refining the model quality at the master. Via extensive experiments using several real-world datasets on AWS Lambda, we show that LocalNewton requires fewer than 60% of the communication rounds (between master and workers) and less than 40% of the end-to-end running time, compared to state-of-the-art algorithms, to reach the same training loss.

As discussed earlier, the low-quality machines in serverless are limited by local memory, which makes it impossible to run training algorithms with a large model dimension (e.g., AWS Lambda workers have a maximum memory of 10 GB). However, in many scenarios, it is only a subset of the features that are most predictive of the outputs storing which requires only a *sublinear memory* in the dimensionality of the data. Feature hashing (FH) is one of the most popular algorithms [68] which uses a universal hash function to project the features. While FH is ideal for prediction, it is not suited for feature selection; that is, the original important features cannot be recovered from the hashed ones. Recently, first-order stochastic gradient descent (SGD) algorithms [69], [70] have been developed which extend the ideas in feature hashing (FH) [68] to feature selection. Instead of explicitly storing the feature vectors, these algorithms store a low-dimensional sketch of the features

in a data structure called Count Sketch [71], originated from the streaming literature.

In Chapter 6, we consider feature selection for applications in machine learning where the dimensionality of the data is so large that it exceeds the working memory of the serverless machine. Unfortunately, current large-scale sketching algorithms show poor memory-accuracy trade-off due to the irreversible collision and accumulation of the stochastic gradient noise in the sketched domain. To mitigate this, we develop a second-order ultra-high dimensional feature selection algorithm, called BEAR, which stores the *second-order stochastic gradients* in the celebrated Broyden Fletcher Goldfarb Shannon (BFGS) algorithm using a Count Sketch, a sublinear memory data structure from the streaming literature. Experiments on real-world data sets demonstrate that BEAR requires *up to three orders of magnitude* less memory space to achieve the same classification accuracy compared to the first-order sketching algorithms. Theoretical analysis proves convergence of BEAR with $\mathcal{O}(1/t)$ rate in t iterations of the sketched algorithm. Our algorithm reveals an unexplored advantage of second-order optimization for memory-constrained sketching of models trained on ultra-high dimensional data sets.

1.4 Non-convex Optimization

Part III caters to non-convex optimization, or specifically, a subset of problems popularly known as *deep learning* that involves training large-scale Deep Neural Network (DNN) based models. Note that practical training of such models require special hardware accelerators such as Graphical Processing Units (GPUs), Tensor Processing Units (TPUs), etc. Commercial platforms do not currently offer these specific hardware-based serverless platforms to accelerate DNN training. However, it is expected that such platforms will soon evolve due to the popularity of serverless [9]. In this part, we propose techniques for large-scale DNN training that retain the generalization gap while reducing communication costs. Our techniques can be applied on any distributed platform, including a future serverless platform that caters to deep learning.

Large-batch training is the standard way of increasing the increasing the number of workers for distributed DNN training. Even though the training loss can be reduced more efficiently, there is a maximum batch size after which the resulting model tends to have worse generalization performance [72]–[76]. Methods which use adaptive batch sizes exist [77]–[81]. However, most of these methods are either designed for specific datasets or require extensive hyper-parameter tuning. Furthermore, they ineffectively use the computational resources by reducing the batch size during part of the training.

Stochastic weight averaging (SWA) [82] is a method where models are sampled from the later stages of an SGD training run. When the weights of these models are averaged, they result in a model with much better generalization properties. This strategy is very effective and has been adopted in multiple domains: deep reinforcement learning [83], semi-supervised learning [84], Bayesian inference [85], low-precision training [86].

In Chapter 7, we adapt SWA to accelerate DNN training. Specifically, we propose Stochastic Weight Averaging in Parallel (SWAP), an algorithm to accelerate distributed training of Deep Neural Networks (DNNs) with a large number of data-parallel workers. SWAP uses large mini-batches to

compute an approximate solution quickly and then refines it by averaging the weights of multiple models computed independently and in parallel. The resulting models generalize equally well as those trained with small mini-batches but are produced in a substantially shorter time. We demonstrate the reduction in training time and the good generalization performance of the resulting models on the computer vision datasets CIFAR10, CIFAR100, and ImageNet.

Due to the use of large clusters with powerful machines to train complex DNNs (e.g. BERT-Large [87] with 340M parameters), the distributed training workloads are becoming increasingly communication bound. For this reason, numerous compression schemes have been proposed in the past several years for the data parallel setting (see [88] for a comprehensive survey). These compression schemes come in various forms, such as the following: (i) Quantization, where the number of bits per entry of the communicated vector is reduced (e.g., [89]–[92]); (ii) Sparsification, where only a few entries of the communicated vector are sent (e.g., [93]–[98]); (iii) Statistical techniques such as Randomized Sketching (e.g., [99], [100]); and (iv) Low-rank approximation, which decomposes the vector into low-rank components before communication (e.g., [101]–[104]).

In Chapter 8, we consider hybrid parallelism—a paradigm that further scales DNN training by employing both Data Parallelism (DP) and Model Parallelism (MP) for distributed training of DNNs. We propose a compression framework called Dynamic Communication Thresholding (DCT) for communication-efficient hybrid training. DCT filters the entities to be communicated across the network through a simple hard-thresholding function, allowing only the most relevant information to pass through. For communication efficient DP, DCT compresses the parameter gradients sent to the parameter server during model synchronization. The threshold is updated only once every few thousand iterations to reduce the computational overhead of compression. For communication efficient MP, DCT incorporates a novel technique to compress the activations and gradients sent across the network during the forward and backward propagation, respectively. This is done by identifying and updating only the most relevant neurons of the neural network for each training sample in the data. We evaluate DCT on natural language processing and recommender system models. DCT reduces communication by $100\times$ and $20\times$ during DP and MP, respectively, improving end-to-end training time on industry scale models by 37% without any loss in performance.

1.5 Resource Allocation and Pricing

In part IV, we propose principled yet practical approaches for allocation of serverless resources that take user job requirements such as delay sensitivity into account. Further, a side-product of our allocation scheme is a pricing scheme that is defined by the interplay between the supply and the demand of the market. These schemes borrow ideas from economics and human psychology to maximize user happiness.

The feature of pricing cloud resources based on delay-sensitivity of users has been lacking from most of the current pricing schemes that are being implemented as well as several real-time dynamic pricing schemes that have been proposed in the literature (we refer the readers to [105], [106] and [107] for surveys on existing and proposed pricing schemes for the cloud).

In Chapter 9, we propose a novel scheduler to allocate resources for serverless computing with the help of utility functions to model the delay-sensitivity of customers. The resulting resource allocation scheme is optimal in the sense that it maximizes the aggregate utility of all users across the system, thus maximizing social welfare. Our approach gives rise to a dynamic pricing scheme that is obtained by solving an optimization problem in its dual form. We further develop feedback mechanisms that allow the cloud provider to converge to optimal resource allocation, even when the users' utilities are private and unknown to the service provider. Simulations show that our approach can track market demand and achieve significantly higher social welfare (or, equivalently, cost savings for customers) compared to existing schemes.

Further, although the market-based pricing scheme in Chapter 9 allows us to differentiate between the users and allocate the resources to customers who "want" them the most, or rather who are ready to pay the most, it has a potential problem which is further accentuated when the demand for resources exceeds the supply by a lot. It drives the prices high and discourages some of the smaller customers from participating in such markets. Indeed, the customers come in all sizes. In particular, small business owners particularly rely on cloud computing services as it allows them to off-load the architecture and back-end duties that are critical to maintaining the computing resources and focus on their core business processes [108].

Several service providers use lottery-based allocations to decide which agents to serve next. For example, First-in-First-out (FIFO) is a popular way of allocating warm queues in serverless computing [109]. As a result, all the customers who are willing to pay the minimum price set by the service provider get a shot at receiving the service. Such lottery-based provisions give rise to uncertainties in the execution of the jobs and their delay times. Just as we distinguished between different customers with varying delay-sensitivity in Chapter 9, another important criteria for service requirement is the uncertainty in execution of jobs, and different agents have different preferences towards these uncertainties. In Chapter 10, we capture these varying preferences towards uncertainties and allocate resources optimally via lotteries.

Part I

Speeding up Linear Algebra

In this part of the thesis, we study algorithms for speeding up algorithms for distributed numerical linear algebra in the serverless setting.

- **Chapter 2:** Using error-correcting codes from information and coding theory for exact distributed computation
- **Chapter 3:** Using sketching methods from randomized numerical linear algebra for approximate distributed computation

Chapter 2

Codes for Serverless Straggler Mitigation

In this chapter, we propose and implement simple yet principled approaches for straggler mitigation in serverless systems for matrix multiplication and evaluate them on several common applications from machine learning and high-performance computing.

2.1 Introduction

To formalize this problem, we consider the typical workflow of a serverless system for the task of matrix-matrix multiplication (see Fig. 2.1). First, worker machines read the input data from the cloud, jointly encode the data, and write the encoded data to the cloud (T_{enc}). Then, the workers start working on their tasks using the encoded data, and write back the product of coded matrices back to the cloud memory. Denote the joint compute time (including the time to communicate the task results to the cloud) T_{comp}. Once a *decodable* set of task results are collected, the workers start running the decoding algorithm to obtain the final output (which takes T_{dec} time). Note that all of these phases are susceptible to straggling workers. Hence, one can write the total execution time of a coded computing algorithm as $T_{\text{tot,coded}} = T_{\text{enc}} + T_{comp} + T_{\text{dec}}$. The key question that we ask is how to minimize end-to-end latency, $T_{\text{tot,coded}}$, that comprises encoding, decoding and computation times, where all of these phases are performed in parallel by serverless workers.

2.1.1 Main Contribution

In this chapter, we advocate principled, coding-based approaches to accelerate distributed computation in serverless computing. Our goals span both theory and practice: we develop coding-based techniques to solve common machine learning problems on serverless platforms in a fault/straggler resilient manner, analyze their runtime and straggler tolerance, and implement them on AWS Lambda for several popular applications.

Generally, computations underlying several linear algebra and optimization problems tend to be iterative in nature. With this in mind, we aim to develop general coding-based approaches for straggler-resilient computation which meet the following criteria: (1) Encoding over big datasets

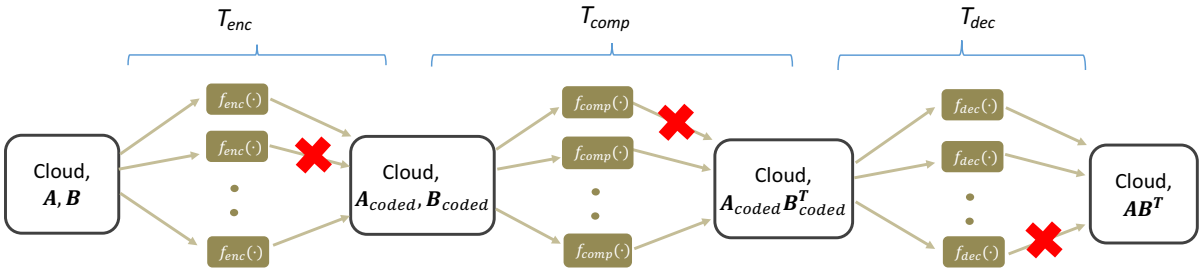


Figure 2.1: Typical workflow on a serverless system for computing the matrix multiplication $\mathbf{AB}^T$. Here, f_{enc}, f_{comp} and f_{dec} denote the functions corresponding to encoding, computation, and decoding, respectively, that are employed at the serverless workers (in parallel on different data points). Whereas most existing schemes focus on minimizing time required to compute the product (T_{comp}), our focus is on minimizing the end-to-end latency that involves parallel encoding (T_{enc}) and decoding (T_{dec}) times as well.

should be performed once. In particular, the cost for encoding the data for straggler-resilient computation will be amortized over iterations. (2) Encoding and decoding should be low-complexity and require at most linear time and space in the size of the data. (3) Encoding and decoding should be amenable to a parallel implementation. This final point is particularly important when working with large datasets on serverless systems due to the massive scale of worker nodes and high communication latency.

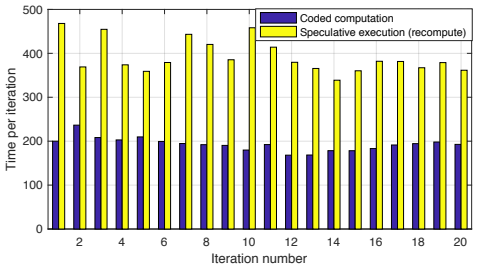
It is unlikely that there is a “one-size-fits-all" methodology which meets the above criteria and introduces straggler resilience for any problem of interest. Hence, we propose to focus our efforts on a few fundamental operations including matrix-matrix multiplication and matrix-vector multiplication, since these form atomic operations for many large-scale computing tasks. Our developed algorithms outperform speculative execution and other popular coding-based straggler mitigation schemes by at least 25%. We demonstrate the advantages of using the developed coding techniques on several applications such as alternating least squares, SVD, Kernel Ridge Regression, power iteration, etc.

2.2 Straggler Resilience in Serverless Computing Using Codes

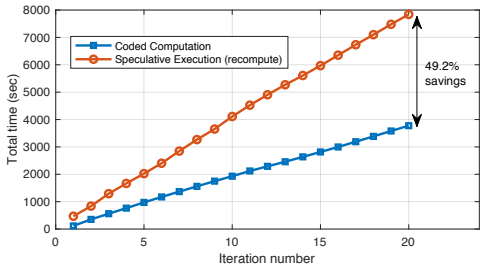
In this section, we describe our schemes for straggler-resilient matrix-vector and matrix-matrix multiplication in the serverless framework.

2.2.1 Distributed Matrix-Vector Multiplication

The main objective of this section is to show that coding schemes can hugely benefit serverless computing by implementing coded matrix-vector multiplication on AWS Lambda. Computing $\mathbf{y} = \mathbf{Ax}$, for a large matrix $\mathbf{A}$, is a frequent bottleneck of several popular iterative algorithms such as gradient descent, conjugate gradient, power iteration, etc. Many coding theory based techniques for straggler-resilient matrix vector multiplication have been proposed in the literature (e.g. see [24], [27], [35], [38]). We refer the reader to Fig. 2 in [24] for an illustration. Fortunately, many of



(a) Per iteration time for power iteration



(b) Running time for power iteration for 20 iterations

Figure 2.2: Coded computing versus speculative execution for power iteration on a matrix of dimension 0.5 million for 20 iterations.

these schemes can be directly employed in serverless systems since the encoding can be done in parallel and the decoding over the resultant output for computing $\mathbf{y}$ is inexpensive as it is performed over a vector. Note that a direct applicability is not true for all operations (such as matrix-matrix multiplication), as we will see later in Section 2.2.2.

To illustrate the advantages of coding techniques over speculative execution, we implement power iteration on the serverless platform AWS Lambda. Power iteration requires a matrix-vector multiplication in each iteration and gives the dominant eigenvector and corresponding eigenvalue of the matrix being considered. Power iteration constitutes an important component for several popular algorithms such as PageRank and Principal Component Analysis (PCA). PageRank is used by Google to rank documents in their search engine [110] and by Twitter to generate recommendations of who to follow [111]. PCA is commonly employed as a means of dimensionality reduction in applications like data visualization, data compression and noise reduction [112].

We applied power iteration to a square matrix of dimension $(0.5 \text{ million})^2$ using 500 workers on AWS Lambda in the Pywren framework [1]. A comparison of compute times of coded computing with speculative execution is shown in Fig. 2.2, where a $2\times$ speedup is achieved[1]. Apart from being significantly faster than speculative execution, another feature of coded computing is reliability, that is, almost all the iterations take a similar amount of time ($\sim$200 seconds) compared to speculative execution, the time for which varies between 340 and 470 seconds. We demonstrate this feature of coded computing throughout our experiments in this chapter.

2.2.2 Distributed Matrix-Matrix Multiplication

Large-scale matrix-matrix multiplication is a frequent computational bottleneck in several problems in machine learning and high-performance computing and has received significant attention from

[1]For our experiments on matrix-vector multiplication, we used the coding scheme proposed in [27] due to its simple encoding and decoding that takes linear time. However, we observed that using other coding schemes that are similar, such as the one proposed in [24], result in similar runtimes.

the coding theory community (e.g. see [26]–[32]). The problem is computing

$$\mathbf{A}\mathbf{B}^T = \mathbf{C}, \text{ where } \mathbf{A} \in \mathbb{R}^{m \times n} \text{ and } \mathbf{B} \in \mathbb{R}^{\ell \times n}. \tag{2.1}$$

Proposed Coding Scheme: For straggler-resilient matrix multiplication, we describe our easy-to-implement coding scheme below. First, we encode the row-blocks of $\mathbf{A}$ and $\mathbf{B}$ in parallel by inserting a parity block after every L_A and L_B blocks of $\mathbf{A}$ and $\mathbf{B}$, respectively, where L_A and L_B are parameters chosen to control the amount of redundancy the code introduces. This produces encoded matrices $\mathbf{A}_{\text{coded}}$ and $\mathbf{B}_{\text{coded}}$. As L_A and L_B are increased, the parity blocks become more spread out, and the code has less redundancy. For example, when $L_A = L_B = 1$, every row of the matrices $\mathbf{A}$ and $\mathbf{B}$ is duplicated (and, hence, has 100% redundancy). At the other extreme, when L_A and L_B are set equal to the number of row-blocks in $\mathbf{A}$ and $\mathbf{B}$, respectively, there is only one parity row-block added in $\mathbf{A}$ and $\mathbf{B}$, and thus, the code exhibits minimum possible redundancy. In Fig. 2.3, an example of the encoded matrix $\mathbf{A}_{\text{coded}}$ and the resultant output matrix $\mathbf{C}_{\text{coded}}$ is shown for the case when $\mathbf{A} = \mathbf{B}$ and $L_A = 2$.

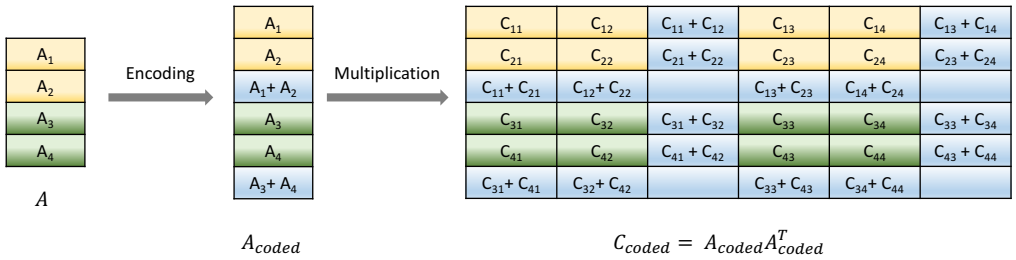


Figure 2.3: Computing $\mathbf{C} = \mathbf{A}\mathbf{A}^T$ where $\mathbf{A}$ is divided into four row-blocks and $L_A = 2$. Here, $\mathbf{C}_{ij} = \mathbf{A}_i\mathbf{A}_j^T$. Locally encoding the rows of $\mathbf{A}$ leads to a locally recoverable code in the output $\mathbf{C}_{\text{coded}}$.

Note the locally recoverable structure of $\mathbf{C}_{\text{coded}}$: to decode one straggler, only a subset of blocks of $\mathbf{C}_{\text{coded}}$ need to be read. In Fig. 2.3, for example, only two blocks need to be read to mitigate a straggler. This is unlike polynomial codes which are MDS in nature and, hence, are optimal in terms of recovery threshold but require reading all the blocks from the output matrix while decoding. The locally recoverable structure of the code makes it particularly amenable to a parallel decoding approach: $\mathbf{C}_{\text{coded}}$ consists of $(L_A + 1) \times (L_B + 1)$ submatrices, each of which can be separately decoded in parallel. In Fig. 2.3, there are four such submatrices. We use a simple peeling decoder (for example, see [26], [27]) to recover the systematic part of each $(L_A + 1) \times (L_B + 1)$ submatrix, constructing the final result matrix $\mathbf{C}$ from these systematic results.

In the event that any of the submatrices are not decodable due to a large number of stragglers, we recompute the straggling outputs. Thus, choosing L_A and L_B presents a trade-off. We would like to keep them small so that we can mitigate more stragglers without having to recompute, but smaller L_A and L_B imply more redundancy in computation and is potentially more expensive. For example, $L_A = L_B = 5$ implies 44% redundancy. Later, we will show how to choose the parameters L_A and L_B given an upper bound on the probability of encountering a straggler in the serverless system. We will also prove that with the right parameters, the probability of not being able to decode the missing blocks is negligible.

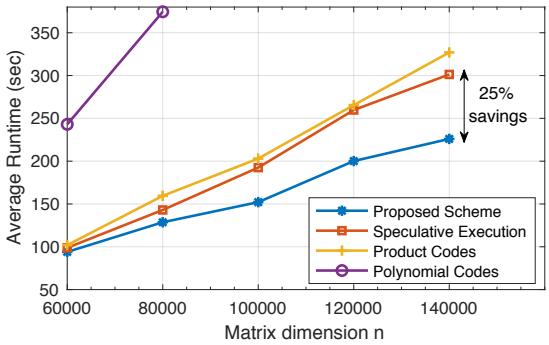


Figure 2.4: Comparison of average runtimes of proposed schemes versus existing schemes for multiplication of two square matrices. For large matrix dimensions, decoding with a polynomial code is not feasible since the master node cannot store all the data locally.

We refer to the proposed coding scheme in Fig. 2.3 as the *local product code*. In Fig. 2.4, we compare the local product code with speculative execution, and existing popular techniques for coded matrix multiplication such as polynomial codes [28] and product codes [26]. In our experiment, we set $\mathbf{A}$ $(= \mathbf{B})$ to be a square matrix with $L_A = 10$, implying 21% redundancy. Product codes and polynomial codes were also designed such that the amount of redundancy was $\geq 21\%$. Accordingly, we wait for 79% of the workers to return before starting to recompute in the speculative execution-based approach so that all the methods employed had the same amount of redundancy. We note that the coding-based approach performs significantly better than existing coding-based schemes and at least 25% better than the speculative execution-based approach for large matrix dimensions[2].

Another important point to note is that existing coding-based approaches perform worse than speculative execution. This is because of the decoding overhead of such schemes. Product codes have to read the entire column (or row) block of $\mathbf{C}_{\text{coded}}$ and polynomial codes have to read the entire output $\mathbf{C}_{\text{coded}}$ to decode one straggler. In serverless systems, where workers write their output to a cloud storage and do not communicate directly with the master owing to their 'stateless' nature, this results in a huge communication overhead. In fact, for polynomial codes, we are not even able to store the entire output in the memory of the master for larger values of n. For this reason, we do not have any global parities—that require reading all the blocks to decode the stragglers—in the proposed local product code. Note that existing coding schemes with locality, such as [27] and [31], also have global parities which are dispensable in serverless and, thus, have high redundancy. This is because such schemes were designed for serverful systems where the decoding is not fully distributed. Moreover, we show in the next section that local product codes are asymptotically optimal in terms of locality for a fixed amount of redundancy. In the event the output is not locally decodable in local product codes, we restart the jobs of straggling workers. However, we later show that such an event is unlikely if the parameters L_A and L_B are chosen properly.

[2]A working implementation of the proposed schemes is available at https://github.com/vvipgupta/serverless-straggler-mitigation

Remark 1. To mitigate stragglers during encoding and decoding phases, we employ speculative execution. However, in our experiments, we have observed that encoding and decoding times have negligible variance and do not generally suffer from stragglers. This is because the number of workers required during encoding and decoding phases is relatively small (less than 10% of the computation phase) with smaller job times due to locality. The probability of encountering a straggler in such small-scale jobs is extremely low.

Remark 2. It has been well established in the literature that blocked partitioning of matrices is communication efficient for distributed matrix-matrix multiplication both in the serverful [113], [114] and serverless [5] settings. Even though in Fig. 2.3 we show partitioning of $\mathbf{A}$ into row-blocks for clarity of exposition, we further partition the input matrices $\mathbf{A}$ (and $\mathbf{B}$) into square blocks in all our experiments and perform block-wise distributed multiplication.

2.3 Theoretical Analysis of Local Product Codes

In this chapter, we analyze Local Product Codes, a coding-theoretic scheme for matrix-matrix multiplication in the serverless framework.

2.3.1 Optimality of Local Product Codes

In coding-theoretic terminology, a locally recoverable code (LRC) is a code where each symbol is a function of small number of other symbols. This number is referred to as the locality, r, of the code. In the context of straggler mitigation, this means that each block in $\mathbf{C}_{\text{coded}}$ is a function of only a few other blocks. Hence, to decode one straggler, one needs to read only r blocks. In the example of Fig. 2.3, the locality is $r = 2$ since each block of $\mathbf{C}_{\text{coded}}$ can be recovered from two other blocks. In general, the locality of the local product code is $\min(L_A, L_B)$. Another important parameter of a code is its minimum distance, d, which relates directly to the number of stragglers that can be recovered in the worst case. Specifically, to recover the data of e stragglers in the worst case, the minimum distance must satisfy $d \geq e + 1$.

For a fixed redundancy, Maximum Distance Separable (MDS) codes attain the largest possible minimum distance d, and thus, are able to tolerate the most stragglers in the worst case. Many straggler mitigation schemes are focused on MDS codes and have gained significant attention, such as polynomial codes [28]. However, such schemes are not practical in the serverless case since they ignore the encoding and decoding costs. Moreover, as seen from Fig. 2.4, it is better to restart the straggling jobs than to use the parities from polynomial or product codes since the communication overhead during decoding is high.

Hence, in serverless systems, the locality r of the code is of greater importance since it determines the time required to decode a straggler. For any LRC code, the following relation between d and r is satisfied [115], [116]

$$d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2, \tag{2.2}$$

where k is the number of systematic data blocks and n is the total number of data blocks including parities. Now, since we want to tolerate at least one straggler, the minimum distance must satisfy $d \geq 2$. Using $\lceil k/r \rceil \geq k/r$, we conclude that $n - k - \frac{k}{r} \geq 0$ or, equivalently,

$$r \geq \frac{k}{n-k}. \tag{2.3}$$

Now, in the case of the local product code, each of the submatrices that can be decoded in parallel represent a product code with $k = L_A L_B$ and $n = (L_A + 1)(L_B + 1)$. In Fig. 2.3, there are four locally decodable submatrices with $L_A = L_B = 2, k = 4$ and $n = 9$. Also, we know that the locality for each of the submatrices is $\min(L_A, L_B)$ and hence this is the locality for the local product code.

Next, we want to compare the locality of the local product code with any other coding scheme with the same parameters, that is, $k = L_A L_B$ and $n = (L_A + 1)(L_B + 1)$. Using Eq. 2.3, we get

$$\begin{aligned} r &\geq \frac{L_A L_B}{(L_A+1)(L_B+1) - L_A L_B} = \frac{L_A L_B}{L_A + L_B + 1} \\ &\geq \frac{\min(L_A, L_B)}{2 + o(1)}. \end{aligned}$$

Thus the locality of local product codes is optimal (within a constant factor) since it achieves the lower bound of locality r for all LRC codes. This is asymptotically better than, say, a local version of polynomial codes (that is, each submatrix of $\mathbf{C}_{\text{coded}}$ is a polynomial code instead of a product code) for which the locality is $L_A L_B$ since it needs to read all $L_A L_B$ blocks to mitigate one straggler [28].

Having shown that local product codes are asymptotically optimal in terms of decoding time, we further quantify the decoding time in the serverless case through probabilistic analysis next.

2.3.2 Decoding Costs

Stragglers arise due to system noise which is beyond the control of the user (and maybe even the cloud provider, for example, unexpected network latency or congestion due to a large number of users). However, a good estimate for an upper bound on the number of stragglers can be obtained through multiple experiments. In our theoretical analysis, we assume that the probability of a given worker straggling is fixed as p, and that this happens independently of other workers. In AWS Lambda, for example, we obtain an upper bound on the number of stragglers through multiple trial runs and observe that less than 2% of the nodes straggle in most trials (also noted from Fig. 1.2). Thus, a conservative estimate of $p = 0.02$ is assumed for AWS Lambda.

Given the high communication latency in serverless systems, codes with low I/O overhead are highly desirable, making locally recoverable codes a natural fit. For local product codes, say the decoding worker operates on a grid of $n = (L_A + 1) \times (L_B + 1)$ blocks. If a decoding worker sees a single straggler, it reads $\min(L_A, L_B)$ blocks to recover it. However, when there are more than one stragglers, at most $L = \max(L_A, L_B)$ block reads will occur per straggler during recovery.

For example, if $L_A > L_B$ and there are two stragglers in the same row, the decoding worker read L_A rows per straggler. Thus, if a decoding worker gets S stragglers, a total of at most SL block reads will occur—there are at most L block reads for each of the S stragglers. Since the number of stragglers, S, is random, the number of blocks read, say R, is also random. Note that R scales linearly with the communication costs.

In Theorem 1, we quantify the decoding costs for local product codes; specifically, we show that the probability of a decoding worker reading a large number of blocks is small.

Theorem 1. *Let p be the probability that a serverless worker straggles independently of others, and R be the number of blocks read by a decoding worker working on $n = (L_A + 1)(L_B + 1)$ blocks. Also, let $L = \max(L_A, L_B)$. Then, the probability that the decoding worker has to read more than x blocks is upper bounded by*

$$\Pr(R \geq x) \leq \left(\frac{x}{npL}\right)^{-x/L} e^{-\frac{x}{L}+np}$$

Proof. See Section 2.5.1. □

Theorem 1 provides a useful insight about the performance of local product codes: the probability of reading more than x blocks during decoding decays decays to zero at a super-exponential rate. Note that for the special (and more practical) case of $L_A = L_B = L$, the number of blocks read per straggler is exactly L and thus $\mathbb{E}[R] = \mathbb{E}[SL] = npL$. Thus, using Theorem 1, we can obtain the following corollary.

Corollary 1. *For any $\epsilon > 0$ and $L = L_A = L_B$, the probability that the decoding worker reads ϵL more blocks than the expected $\mathbb{E}[R]$ blocks is upper bounded by*

$$\Pr(R \geq \mathbb{E}[R] + \epsilon L) \leq \left(1 + \frac{\epsilon}{np}\right)^{-np-\epsilon} e^{-\epsilon}.$$

For $\epsilon = np$, this becomes

$$\Pr(R \geq 2\mathbb{E}[R]) \leq \frac{1}{(4e)^{np}}.$$

In Fig. 2.5, we plot the upper bound on $\Pr(R \geq x)$ for different values of x. The values of n and L were chosen to be consistent with the experiments in Fig. 2.4, where $L_A = L_B = 10$, so that the maximum number of blocks read per straggler is $L = 10$ and the number of blocks of $\mathbf{C}_{\text{coded}}$ per decoding worker is $n = 121$. Additionally, we used $p = .02$ as obtained through extensive experiments on AWS Lambda (see Fig. 1.2). In a polynomial code with the same locality, 100 blocks would be read to mitigate any straggler by a decoding worker. For the local product code, the probability that 100 blocks are read is upper bounded by $\Pr(R \geq 100) \leq 3.5 \times 10^{-10}$.

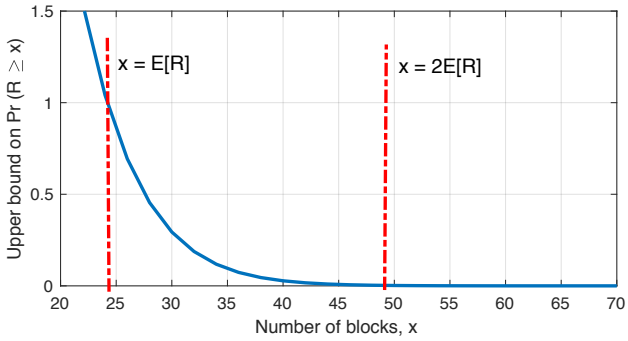


Figure 2.5: Probabilistic upper bound on the number of blocks read, R, by a decoding worker from Theorem 1 shown for $L = 10$, $n = 121$, and $p = 0.02$. Here, $\Pr(R \geq 2\mathbb{E}[R]) \leq 3.1 \times 10^{-3}$.

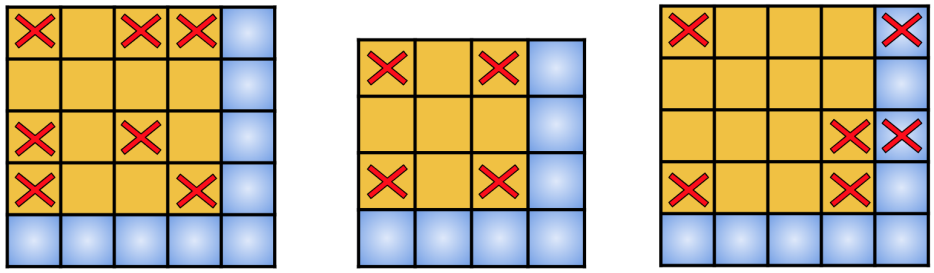


Figure 2.6: Some examples of undecodable sets, as viewed from a single decoding worker's $(L_A + 1) \times (L_B + 1)$ grid. The yellow blocks correspond to the systematic part of the code, and blue blocks to the parity. Blocks marked with an "X" are stragglers.

2.3.3 Straggler Resiliency of Local Product Codes

To characterize the straggler resiliency of local product codes, we turn our focus to finding the probability of encountering an *undecodable set*: a configuration of stragglers that cannot be decoded until more results arrive.

Definition 1. Undecodable set: Consider a single decoding worker that is working on n blocks, arranged in an $(L_A+1) \times (L_B+1)$ grid, and let S be the number of missing workers. The decoding worker's blocks are said to form an S-undecodable set if we need to wait for more workers to arrive to decode all the S missing blocks.

Some examples of undecodable sets are shown in Fig. 2.6. In an S-undecodable set, it is possible that some of the S stragglers are decodable, but there will always be some stragglers that are preventing each other from being decoded. For the local product code, an individual straggler is undecodable if and only if there is at least one other straggler in both its row and column, because the code provides a single redundant block along each axis that can be used for recovery. This implies that a decoding worker must encounter at least three stragglers for one of them to be undecodable. However, the code can always recover any three stragglers through the use of a peeling decoder

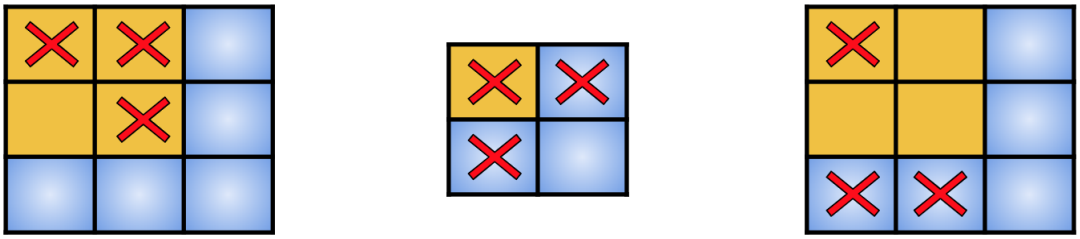


Figure 2.7: Some examples of "interlocking" three straggler configurations. Stragglers can be decoded using a peeling decoder.

[26], [27]. While the three stragglers may share a column or row and be in an "interlocking" configuration, such as those shown in Fig. 2.7, two of the three can always be recovered, or "peeled off". Using these blocks, the straggler that was originally undecodable can be recovered. This provides a key result: all undecodable sets consist of four or more stragglers. Equivalently, given $S \leq 3$, the probability of being unable to decode is zero. This can also be noted directly from the fact the minimum distance of a product code with one parity row and column is four, and hence, it can tolerate any three stragglers [26].

The following theorem bounds the probability of encountering an undecodable set for local product codes.

Theorem 2. *Let p be the probability that a serverless worker straggles independently of others. Let $\bar{D}$ be the event that a decoding worker working on n (≥ 8) blocks in an $(L_A+1) \times (L_B+1)$ grid cannot decode. Then,*

$$\Pr(\bar{D}) \leq \sum_{s=4}^{7} \alpha_s p^s (1-p)^{n-s} + \sum_{s=8}^{n} \binom{n}{s} p^s (1-p)^{n-s},$$

where

$$\alpha_4 = \binom{L_A+1}{2}\binom{L_B+1}{2}, \quad \alpha_5 = \alpha_4(n-4),$$

$$\alpha_6 \leq \binom{L_A+1}{3}\binom{L_B+1}{3}\binom{9}{6} + \alpha_4\binom{n-4}{2}, \text{ and}$$
$$\alpha_7 \leq \binom{L_A+1}{3}\binom{L_B+1}{3}\binom{9}{7} + \alpha_4\binom{n-4}{3}$$

Proof. See Section 2.5.2. □

In Fig. 2.8, the bound in Theorem 2 is shown with $p = 0.02$ for $L = L_A = L_B = 1, 2, ..., 25$ so that the total number of blocks per worker is $(L+1)^2$. This shows a "sweet spot" around 121

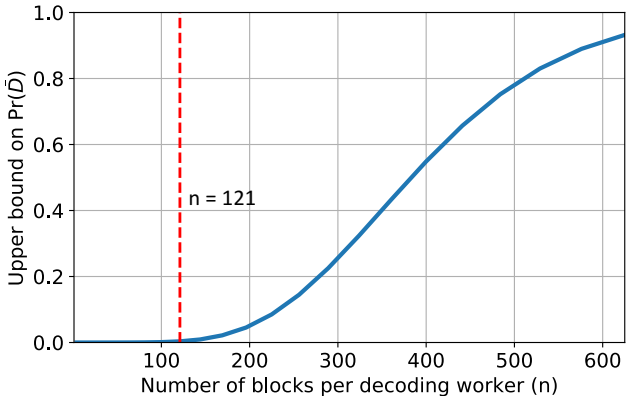


Figure 2.8: Upper bound on probability of the event $\bar{D}$ (that is, a decoding worker being unable to decode) when $p = .02$. We chose $n = 121$ in our experiments which represents a good trade-off between code redundancy and straggler resiliency.

blocks per decoding worker, or $L = 10$, the same choice used in the experiments shown in Fig. 2.4. With this choice of code parameters, the probability of a decoding worker being able to decode all the stragglers is high. This simultaneously enables low encoding and decoding costs, avoids doing too much redundant computation during the multiplication stage (only 21%), and gives a high probability of avoiding an undecodable set in the decoding stage. In particular, for $L_A = L_B = 10$, an individual worker is able to decode with probability at least 99.64% when $p = 0.02$.

Remark 3. The analysis in Sections 2.3.2 and 2.3.3 derives bounds for one decoding worker. In general, for decoding using k workers in parallel, the respective upper bounds on probabilities in Theorem 1 (any decoding worker reading more than x blocks) and Theorem 2 (any decoding worker not able to decode) will be multiplied by k using the union bound.

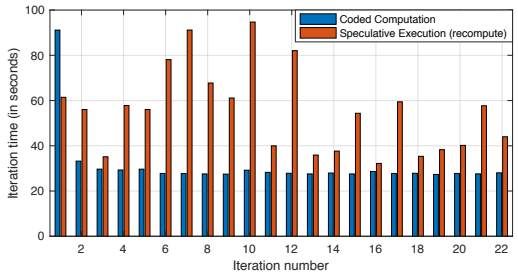
2.4 Coded Computing in Applications

In this section, we take several high-level applications from the field of machine learning and high performance computing, and implement them on the serverless platform AWS Lambda. Our experiments clearly demonstrate the advantages of proposed coding schemes over speculative execution.

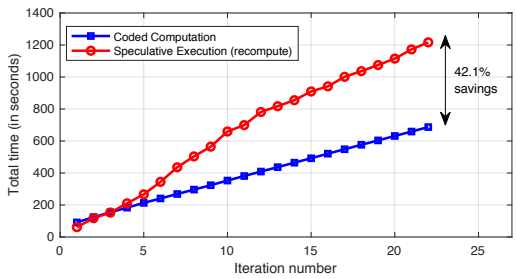
2.4.1 Kernel Ridge Regression

We first focus on the flexible class of Kernel Ridge Regression (KRR) problems with Preconditioned Conjugate Gradient (PCG). Oftentimes, KRR problems are ill-conditioned, so we use a preconditioner described in [117] for faster convergence. The problem can be described as

$$(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{x} = \mathbf{y}, \tag{2.4}$$

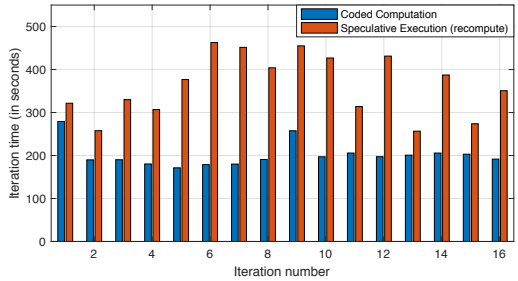


(a) Per iteration time during PCG for ADULT dataset.

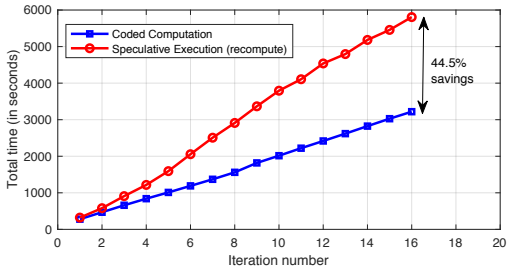


(b) Total running time for PCG for ADULT dataset.

Figure 2.9: Coded computing versus speculative execution for KRR with PCG on the ADULT dataset. Error on testing dataset was 11%.



(a) Per iteration time during PCG for EPSILON dataset



(b) Total running time for PCG for EPSILON dataset

Figure 2.10: Coded computing versus speculative execution for KRR with PCG on the EPSILON dataset. Error on testing dataset was 8%.

where $\mathbf{K} \in \mathbb{R}^{n \times n}$ is a Kernel matrix defined by $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ with the kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ on the input domain $\mathcal{X} \subseteq \mathbb{R}^d$, n is the number of samples in training data, $\mathbf{y} \in \mathbb{R}^{n \times 1}$ is the labels vector and the solution to coefficient vector $\mathbf{x}$ is desired. A preconditioning matrix $\mathbf{M}$ based on random feature maps [118] can be introduced for faster convergence, so that the KRR problem in Eq. (2.4) can be solved using Algorithm 1. Incorporation of such maps has emerged as a powerful technique for speeding up and scaling kernel-based computations, often requiring fewer than 20 iterations of Algorithm 1 to solve (2.4) with good accuracy.

Straggler mitigation with coding theory: The matrix-vector multiplication in Steps 4 and 6 are the bottleneck in each iteration and are distributedly executed on AWS Lambda. As such, they are prone to slowdowns due to faults or stragglers, and should be the target for the introduction of coded computation. To demonstrate the promised gains of the coding theory based approach, we conducted an experiment on the standard classification datasets ADULT and EPSILON [119] with Gaussian kernel $k(\mathbf{x}, \mathbf{z}) = \exp(-||\mathbf{x} - \mathbf{z}||_2^2 / 2\sigma^2)$ with $\sigma = 8$ and $\lambda = 0.01$, and the Kernel matrices are square of dimension $32,000$ and $400,000$, respectively. We store the training and all subsequently generated data in cloud storage S3 and use Pywren [1] as a serverless computing framework on AWS Lambda.

Algorithm 1: Fast Kernel Ridge Regression using preconditioned conjugate gradient

Input Data (stored in S3): Kernel Matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ and vector $\mathbf{y} \in \mathbb{R}^{n \times 1}$, regularization parameter λ, inverse of the preconditioner $\mathbf{M} \in \mathbb{R}^{n \times n}$ found using the random feature map from [118]

Initialization: Define $\mathbf{x}_0 = \mathbf{1}^{n \times 1}$, $\mathbf{r}_0 = \mathbf{y} - (\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{x}_0$, $\mathbf{z}_0 = \mathbf{M}^{-1}\mathbf{r}_0$, $\mathbf{p}_0 = \mathbf{z}_0$

while $\|(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{x}_k - \mathbf{y}\| > 10^{-3}\|\mathbf{y}\|$ **do**

  $\mathbf{h}_k = (\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{p}_k$; // Computed in parallel using codes

  $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{z}_k}{\mathbf{p}_k^T \mathbf{h}_k}$, $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{h}_k$

  $\mathbf{z}_{k+1} = \mathbf{M}^{-1}\mathbf{r}_{k+1}$; // Computed in parallel using codes

  $\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{z}_{k+1}}{\mathbf{r}_k^T \mathbf{z}_k}$, $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$

end

Result: $\mathbf{x}^* = \mathbf{x}_{k+1}$ where $(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{x}^* = y$

For this experiment, we implemented a 2D product code similar to that proposed in [27] to encode the row-blocks of $(\mathbf{K} + \lambda \mathbf{I}_n)$ and $\mathbf{M}^{-1}$, and distributed them among 64 and 400 Lambda workers, respectively. To compare this coded scheme's performance against speculative execution, we distribute the uncoded row-blocks of $(\mathbf{K} + \lambda \mathbf{I}_n)$ and $\mathbf{M}^{-1}$ among the same number of Lambda workers, and wait for 90% of jobs to finish and restart the rest without terminating unfinished jobs. Any job that finishes first would submit its results. The computation times for KRR with PCG on these datasets for the coding-based and speculative execution-based schemes is plotted in Figs. 2.9 and 2.10. For coded computation, the first iteration also includes the encoding time. We note that coded computation performs significantly better than speculative execution, with 42.1% and 44.5% reduction in total job times for ADULT and EPSILON datasets, respectively. This experiment again demonstrates that coding-based schemes can significantly improve the efficiency of large-scale distributed computations. Other regression problems such as ridge regression, lasso, elastic net and support vector machines can be modified to incorporate codes in a similar fashion.

2.4.2 Alternating Least Squares

Alternating Least Squares (ALS) is a widely popular method to find low rank matrices that best fit the given data. This empirically successful approach is commonly employed in applications such as matrix completion and matrix sensing used to build recommender systems [120]. For example, it was a major component of the winning entry in the Netflix Challenge where the objective was to predict user ratings from already available datasets [121]. We implement the ALS algorithm for matrix completion on AWS Lambda using the Pywren framework [1], where the main computational bottleneck is a large matrix-matrix multiplication in each iteration.

Let $\mathbf{R} \in \mathbb{R}^{u \times i}$ be a matrix constructed based on the existing (incomplete) ratings, where u and i are the number of users giving ratings and items being rated, respectively. The objective is to find the matrix $\tilde{\mathbf{R}}$ which predicts the missing ratings. One solution is to compute a low-rank

Algorithm 2: Alternating Least Squares (ALS)

Input Data (stored in S3): Ratings Matrix $\mathbf{R} \in \mathbb{R}^{u \times i}$, regularization parameter λ, latent factor dimension f, desired accuracy ϵ

Initialization: Define $\mathbf{H}_0 \in \mathbb{R}^{u \times f}$, $\mathbf{W}_0 \in \mathbb{R}^{f \times i}$ with entries drawn independently from a Uniform$[0, 1/f]$ distribution.

while $||\mathbf{R} - \mathbf{H}_k\mathbf{W}_k||_F^2 > \epsilon$ **do**

  User step: $\mathbf{H}_k = \mathbf{R}\mathbf{W}_{k-1}^T(\mathbf{W}_{k-1}\mathbf{W}_{k-1}^T + \lambda\mathbf{I}_f)^{-1}$; `// Done in parallel using codes`

  Item step: $\mathbf{W}_k = (\mathbf{H}_k^T\mathbf{H}_k + \lambda\mathbf{I}_f)^{-1}\mathbf{H}_k^T\mathbf{R}$; `// Done in parallel using codes`

end

Result: $\mathbf{H}^* = \mathbf{H}_k$, $\mathbf{W}^* = \mathbf{W}_k$

factorization based on the existing data, which decomposes the ratings matrix as $\tilde{\mathbf{R}} = \mathbf{H}\mathbf{W}$, where $\mathbf{H} \in \mathbb{R}^{u \times f}$, $\mathbf{W} \in \mathbb{R}^{f \times i}$ for some number of latent factors f, which is a hyperparameter.

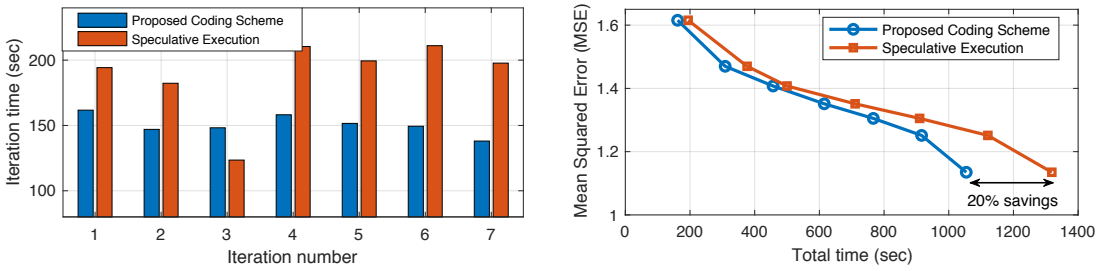
Let us call the matrices $\mathbf{H}$ and $\mathbf{W}$ the *user matrix* and *item matrix*, respectively. Each row of $\mathbf{H}$ and column of $\mathbf{W}$ uses an f-dimensional vector of latent factors to describe each user or item, respectively. This gives us a rank-f approximation to $\mathbf{R}$. To obtain the user and item matrices, we solve the optimization problem $\text{argmin}_{\mathbf{H},\mathbf{W}} F(\mathbf{H}, \mathbf{W})$, where the loss $F(\mathbf{H}, \mathbf{W})$ is defined as

$$F(\mathbf{H}, \mathbf{W}) = ||\mathbf{R} - \tilde{\mathbf{R}}||_F^2 + \lambda(||\mathbf{H}||_F^2 + ||\mathbf{W}||_F^2),$$

where $\lambda > 0$ is a regularization hyperparameter chosen to avoid overfitting. The above problem is non-convex in general. However, it is *bi-convex*—given a fixed $\mathbf{H}$, it is convex in $\mathbf{W}$, and given a fixed $\mathbf{W}$, it is convex in $\mathbf{H}$. ALS, described in Algorithm 2, exploits this bi-convexity to solve the problem using coordinate descent. ALS begins with a random initialization of the user and item matrices. It then alternates between a user step, where it optimizes over the user matrix using the current item matrix estimate, and an item step, optimizing over the item matrix using the newly obtained user matrix. Thus, the updates to the user and item matrices in the k-th iteration are given by

$$\begin{aligned}\mathbf{H}_k &= \underset{\mathbf{H}}{\text{argmin}}\, F(\mathbf{H}, \mathbf{W}_{k-1}) \\ &= \mathbf{R}\mathbf{W}_{k-1}^T(\mathbf{W}_{k-1}\mathbf{W}_{k-1}^T + \lambda\mathbf{I}_f)^{-1}; \\ \mathbf{W}_k &= \underset{\mathbf{W}}{\text{argmin}}\, F(\mathbf{H}_k, \mathbf{W}) = (\mathbf{H}_k^T\mathbf{H}_k + \lambda\mathbf{I}_f)^{-1}\mathbf{H}_k^T\mathbf{R}.\end{aligned}$$

In practice, $u, i \gg f$, so computing and inverting the $f \times f$ matrix in each step can be done locally at the master node. Instead, the matrix multiplications $\mathbf{R}\mathbf{W}_{k-1}^T$ and $\mathbf{R}^T\mathbf{H}_k$ in the user and item steps, respectively, are the bottleneck in each iteration, requiring $\mathcal{O}(uif)$ time. To mitigate stragglers, we use local product codes and speculative execution and compare their runtimes in Fig. 2.11 for seven iterations. The matrix $\mathbf{R}$ was synthetically generated with $u = i = 102400$



(a) Per iteration time for ALS.

(b) Total running time versus mean squared error for ALS.

Figure 2.11: Comparison of proposed coding scheme, that is, local product codes, versus speculative execution for straggler mitigation on AWS Lambda.

and the number of latent factors used was $f = 20480$. Each rating was generated independently by sampling a Uniform$\{1, 2, 3, 4, 5\}$ random variable, intended to be the true user rating. Then, noise generated by sampling a $\mathcal{N}(0, .2)$ distribution was added, and the final rating was obtained by rounding to the nearest integer. The ratings matrix $\mathbf{R}$ is encoded once before the computation starts, and thus the encoding cost is amortized over iterations. We used 500 workers during the computation phase and 5 workers during the decoding phase for each matrix multiplication. It can be seen that codes perform 20% better than speculative execution while providing reliability, that is, each iteration takes on average $\sim$150 seconds with much smaller variance in running times per iteration.

2.4.3 Tall-Skinny SVD

Singular Value Decomposition (SVD) is a common numerical linear algebra technique with numerous applications, such as in the fields of image processing [122], genomic signal processing [123], unsupervised learning [112], and more. In this section, we employ our proposed coding scheme in mitigating stragglers while computing the SVD of a tall, skinny matrix $\mathbf{A} \in \mathbb{R}^{m \times p}$, where $m \gg p$. That is, we would like to compute the orthogonal matrices $\mathbf{U} \in \mathbb{R}^{m \times p}$ and $\mathbf{V} \in \mathbb{R}^{p \times p}$ and the diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{p \times p}$, where $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.

To this end, we first compute the matrix-matrix multiplication $\mathbf{B} = \mathbf{A}^T\mathbf{A}$ which is the main computational bottleneck and requires $\mathcal{O}(mp^2)$ time. Next, we compute the SVD of $\mathbf{B}$. Note that $\mathbf{B} \in \mathbb{R}^{p \times p}$ is a smaller matrix and its SVD $\mathbf{B} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$ requires only $O(p^3)$ time and memory and can be computed locally at the master node in general. This will give us the matrix $\mathbf{V}$ and the diagonal matrix $\mathbf{\Sigma}$. Now, $\mathbf{U}$ can again be computed in parallel using the matrix-matrix multiplication $\mathbf{U} = \mathbf{A} \times (\mathbf{V}\mathbf{\Sigma}^{-1})$ which requires $\mathcal{O}(mp^2)$ time.

We compute the SVD of a tall matrix of size $300,000 \times 30,000$ on AWS Lambda. For local product codes, we use 400 systematic workers during computation with 21% redundancy, and 20 and 4 workers for parallel encoding and decoding, respectively. For speculative execution, we employed 400 workers for computing in the first phase and started the second phase (that is, recomputing the straggling nodes) as soon as 79% of the workers from the first phase arrive. Averaged over 5

trials, coded computing took 270.9 seconds compared to 368.75 seconds required by speculative execution, thus providing a 26.5% reduction in end-to-end latency.

Though we do not implement it here, Cholesky decomposition is yet another application that uses matrix-matrix multiplication as an important constituent. It is frequently used in finding a numerical solution of partial differential equations [124], solving optimization problems using quasi-Newton methods [125], Monte Carlo methods [126], Kalman filtering [127], etc. The main bottleneck in distributed Cholesky decomposition involves a sequence of large-scale outer products [3], [17] and hence local product codes can be readily applied to mitigate stragglers.

2.5 Proofs

2.5.1 Proof of Theorem 1

To prove Theorem 1, we use a standard Chernoff bound argument. In particular, for any $t > 0$, we can upper bound the probability of reading at least x blocks as

$$\Pr(R \geq x) \leq e^{-tx} M_R(t), \tag{2.5}$$

where $M_R(t) := \mathbb{E}\left[e^{tR}\right]$ is the Moment Generating Function (MGF) of the random variable R.

We know that the number of blocks read, $R \leq SL$ since we read $\leq L$ blocks every time we decode a straggler. Thus, we can bound $M_R(t)$, the MGF of R, in terms of the MGF of S, $M_S(\tau) = \mathbb{E}\left[e^{\tau S}\right]$, as

$$M_R(t) = \mathbb{E}\left[e^{tR}\right] \leq \mathbb{E}\left[e^{tLS}\right] = M_S(\tau)|_{\tau=tL} \ \ \forall\, t > 0. \tag{2.6}$$

Since we assume each worker straggles independently with probability p, the distribution of S is Binomial(n, p). Thus, its moment generating function is $M_S(\tau) = (1 - p + pe^{\tau})^n$. Using Eq. 2.6, we have $M_R(t) \leq \left(1 - p + pe^{tL}\right)^n$. Using this inequality and the fact that $1 - y \leq e^{-y} \ \ \forall\, y \in \mathbb{R}$ in the upper bound of Eq. 2.5, we get

$$\Pr(R \geq x) \leq e^{-tx + np - np(\exp(tL))} \ \ \forall\, t \geq 0. \tag{2.7}$$

As a last step, we specialize by setting $t = \dfrac{1}{L} LocalNewton\left(\dfrac{x}{npL}\right)$, which is obtained by optimizing the RHS above with respect to t. Substitution into Eq. 2.7 gives the desired upper bound on $\Pr(R \geq x)$, proving Theorem 1.

2.5.2 Proof of Theorem 2

We already discussed in Sec. 2.3.3 that local product codes can decode any three stragglers. Now, we turn our attention to the case of four or more stragglers. Regardless of how much redundancy is used—including the extreme case of $L_A = L_B = 1$ where every block is duplicated three times—there exist undecodable sets with four stragglers. An example is shown in the middle figure

in Fig. 2.6. All 4-undecodable sets come in squares, with every straggler blocking another two off (otherwise, one would be free and decodable, reducing to three stragglers which can always be handled by a peeling decoder). Using this observation, we can create any 4-undecodable set by picking the two rows (from our $L_A + 1$ choices) and two columns (from our $L_B + 1$ choices) to place the stragglers in, yielding exactly four spots. Let α_S be the number of undecodable sets with S stragglers. Thus,

$$\alpha_4 = \binom{L_A + 1}{2}\binom{L_B + 1}{2}.$$

All 5-undecodable sets come in the form of 4-undecodable sets with a fifth straggler placed in any vacant spot on the grid. This gives us a method to count the number of 5-undecodable sets. First, choose the two rows and two columns that make up the embedded 4-undecodable set. Then, choose from any of the $n - 4$ vacant entries to place the fifth straggler, which gives $\alpha_5 = \binom{L_A+1}{2}\binom{L_B+1}{2}(n - 4)$.

In the case of $S = 6, 7$, undecodable sets can be formed in one of two ways: confining all stragglers to three rows and three columns, or constructing a 4-undecodable set and then placing two (or three for $S = 7$) more stragglers anywhere. We can count the former as

$$\binom{L_A + 1}{3}\binom{L_B + 1}{3}\binom{9}{S} \tag{2.8}$$

for both $S = 6$ and $S = 7$ since choosing three rows and three columns yields nine blocks, of which we choose S. For the latter, we can first construct a 4-undecodable set by picking the two rows and two columns in which to place the stragglers, and then place the remaining $S - 4$ anywhere else, giving a total of

$$\binom{L_A + 1}{2}\binom{L_B + 1}{2}\binom{n - 4}{S - 4} \tag{2.9}$$

such undecodable sets. By summing Eqs. 2.8 and 2.9, we obtain an upper bound on α_S for $S = 6, 7$. This is an upper bound, rather than the exact number of undecodable sets, due to the fact that all sets are counted, but several are overcounted. For example, any 6-undecodable set where all six stragglers are confined to a contiguous 2×3 grid is counted by both terms.

In general, if there are S stragglers, there are $\binom{n}{S}$ ways to arrange the stragglers. Given the number of stragglers S, all configurations are equally likely, and the probability of being unable to decode is the percentage of configurations that are undecodable sets. Since $\{\alpha_S\}_{S=4}^{7}$ is the number of S-undecodable sets, the probability of being unable to decode given $S(= 4, 5, 6, 7)$ stragglers is $\binom{n}{S}^{-1}\alpha_S$.

The probability of encountering eight or more stragglers is small for suitably chosen L_A, L_B, owing to the fact that the probability of encountering a straggler is small (for example, $p \approx .02$ for AWS Lambda). Accordingly, we have chosen to focus our analysis on determining α_S for $S \leq 7$. We can obtain an upper bound on the probability of being unable to decode by assuming all configurations where $S \geq 8$ are undecodable sets. Let $\bar{D}$ denote the event that a decoding worker

cannot decode. Then by the law of total probability,

$$\begin{aligned}\Pr(\bar{D}) &= \sum_{s=0}^{n} \Pr(\bar{D}|S=s)\Pr(S=s) \\ &\leq \sum_{s=4}^{7} \binom{n}{s}^{-1} \alpha_s \Pr(S=s) + \sum_{s=8}^{n} \Pr(\bar{D}|S=s)\Pr(S=s).\end{aligned}$$

Now using the inequality $\Pr(\bar{D}|S=s) \leq 1 \; \forall \; s \geq 8$ and $\Pr(S=s) = \binom{n}{s} p^s (1-p)^{n-s}$ gives the desired upper bound, proving Theorem 2.

2.6 Conclusions and Future Work

In this chapter, we argued that in the serverless setting—where communication costs greatly outweigh computation costs—performing some redundant computation based on ideas from coding theory will outperform speculative execution. Moreover, the design of such codes should leverage locality to attain low encoding and decoding costs. Our proposed scheme for coded matrix-matrix multiplication outperforms the widely used method of speculative execution and existing popular coded computing schemes in a serverless computing environment. All three stages of the coded approach are amenable to a parallel implementation, utilizing the dynamic scaling capabilities of serverless platforms. We showed that our proposed scheme is asymptotically optimal in terms of decoding time and further quantified the communication costs during decoding through probabilistic analysis. Additionally, we derived an upper bound on the probability of being unable to decode stragglers.

The proposed schemes for fault/straggler mitigation are *universal* in the sense that they can be applied to many existing algorithms without changing their outcome. This is because they mitigate stragglers by working on low-level steps of the algorithm which are often the computational bottleneck, such as matrix-vector or matrix-matrix multiplication, thus not affecting the algorithm from the application or user perspective. A possible future work is to devise similar schemes for other matrix operations such as distributed QR decomposition, Gaussian elimination, eigenvalue decomposition, etc. Eventually, an influential step would be to create a software library implementing the proposed algorithms for running massive-scale Python code on commercial serverless frameworks. This library would provide a seamless experience for users: they will execute their algorithms on serverless systems (using frameworks such as Pywren [1]) as they normally would, and our algorithms can be automatically invoked “under the hood" to introduce fault/straggler-resilience, thus aligning with the overarching goal of serverless systems to reduce management on the user front.

Chapter 3

Sketch-based Serverless Straggler Mitigation

In this chapter, we propose OverSketch, an approximate algorithm for distributed matrix multiplication in serverless computing. OverSketch leverages ideas from matrix sketching and high-performance computing to enable cost-efficient multiplication that is resilient to faults and straggling nodes pervasive in low-cost serverless architectures.

3.1 Introduction

Matrix multiplication is a frequent computational bottleneck in fields like scientific computing, machine learning, graph processing, etc. In many applications, such matrices are very large, with dimensions easily scaling up to millions. Consequently, the last three decades have witnessed the development of many algorithms for parallel matrix multiplication for High Performance Computing (HPC). During the same period, technological trends like Moore's law made arithmetic operations faster and, as a result, the bottleneck for parallel computation shifted from computation to communication. Today, the cost of moving data between nodes exceeds the cost of arithmetic operations by orders of magnitude, and this gap is increasing exponentially with time [128]–[130]. This has led to the popularity of communication-avoiding algorithms for parallel computation [113], [130].

Large-scale matrix multiplication, being embarrassingly parallel and frequently encountered, is a natural fit for serverless computing. However, existing distributed algorithms for "serverful" systems cannot, in general, be extended to serverless computing due to the following crucial differences between the two architectures:

- Workers in the serverless setting, unlike cluster nodes, do not communicate amongst themselves. They read/write data directly from/to a single data storage entity (for example, cloud storage like AWS S3) and the user is only allowed to submit prespecified jobs and does not have any control over the management of workers [1]–[3].

- Distributed computation in HPC/server-based systems is generally limited by the number of workers at disposal. However, in serverless systems, the number of inexpensive workers can

easily be scaled into the thousands, but these low-commodity nodes are generally limited by the amount of memory and lifespan available.

- Unlike HPC, nodes in the cloud-based systems suffer degradation due to system noise which can be a result of limited availability of shared resources, network latency, hardware failure, etc. [18], [19]. This causes variability in job times, which results in subsets of slower nodes, often called *stragglers*, which significantly slow the computation. Time statistics for worker job times are plotted in Figure 1.2 for AWS Lambda. Notably, there are a few workers ($\sim 2\%$) that take much longer than the median job time, thus decreasing the overall computational efficiency of the system. Distributed algorithms robust to such unreliable nodes are desirable in cloud computing.

3.1.1 Contributions

This chapter bridges the gap between communication-efficient algorithms for distributed computation and existing methods for straggler-resiliency. To this end, we first analyze the monetary cost of distributed matrix multiplication for serverless computing for two different schemes of partitioning and distributing the data. Specifically, we show that row-column partitioning of input matrices requires asymptotically more communication than blocked partitioning for distributed matrix multiplication, similar to the optimal communication-avoiding algorithms in the HPC literature.

In applications like machine learning, where the data itself is noisy, solution accuracy is often traded for computational efficiency. Motivated by this, we propose OverSketch, a sketching scheme to perform blocked *approximate* matrix multiplication and prove statistical guarantees on the accuracy of the result. OverSketch has threefold advantages:

1. Reduced computational complexity by significantly decreasing the dimension of input matrices using sketching,

2. Resiliency against stragglers and faults in serverless computing by over-provisioning the sketch dimension,

3. Communication efficiency for distributed multiplication due to the blocked partition of input matrices.

Sketching for OverSketch requires linear time that is embarrassingly parallel. Through experiments on AWS Lambda, we show that small redundancy ($\approx 5\%$) is enough to tackle stragglers using OverSketch. Furthermore, we use OverSketch to calculate the Hessian distributedly while solving a large linear program using interior point methods and demonstrate a 34% reduction in total compute time on AWS Lambda.

3.1.2 Related Work

Traditionally, techniques like speculative execution are used to deal with stragglers, for example, Hadoop MapReduce [22] and Apache Spark [23]. Such techniques work by detecting nodes that are running slowly or will slow down in the future and then assigning their jobs to new nodes

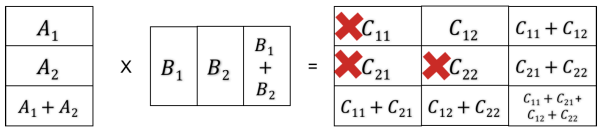


Figure 3.1: Matrix $\mathbf{A}$ is divided into 2 row chunks $\mathbf{A}_1$ and $\mathbf{A}_2$, while $\mathbf{B}$ is divided into two column chunks $\mathbf{B}_1$ and $\mathbf{B}_2$. During the encoding process, redundant chunks $\mathbf{A}_1 + \mathbf{A}_2$ and $\mathbf{B}_1 + \mathbf{B}_2$ are created. To compute $\mathbf{C}$, 9 workers store each possible combination of a chunk of $\mathbf{A}$ and $\mathbf{B}$ and multiply them. During the decoding phase, the master can recover the affected data (C_{11}, C_{12} and C_{22} in this case) using the redundant chunks.

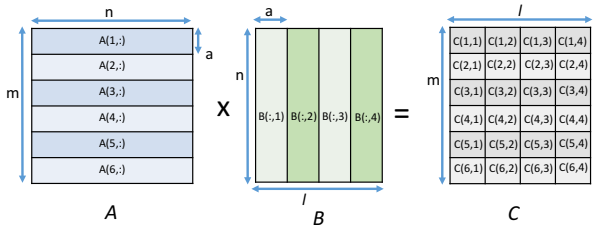
without shutting down the original job. The node that finishes first submits its results. This has many limitations. A constant monitoring of jobs is required, which might be costly if there are many workers in the system. It is also possible that a node will straggle only towards the end of the job, and by the time the job is resubmitted, the additional time and computational overhead has already hurt the overall efficiency of the system. The situation is even worse for smaller jobs, as spinning up an extra node requires additional invocation and setup time which can exceed the job time itself.

Recently, approaches based on coding theory have been developed which cleverly introduce redundancy into the computation to deal with stragglers [24], [26]–[29], [34]. Many of these proposed schemes have been dedicated to distributed matrix multiplication [26]–[29]. In [26], the authors develop a coding scheme for matrix multiplication that uses Maximum Distance Separable (MDS) codes to code $\mathbf{A}$ in a column-wise fashion and $\mathbf{B}$ in a row-wise fashion, so that the resultant is a product-code of $\mathbf{C}$, where $\mathbf{C} = \mathbf{A} \times \mathbf{B}$. An illustration is shown in Figure 3.1. A simpler version of this has been known in the HPC community as Algorithm-Based-Fault-Tolerance (ABFT) [131]. Authors in [27] generalize the results in [26] to a d-dimensional product code with only one parity in each dimension. In [28], the authors develop polynomial codes for matrix multiplication, which is an improvement over [26] in terms of recovery threshold, that is, the minimum number of workers required to recover the product $\mathbf{C}$.

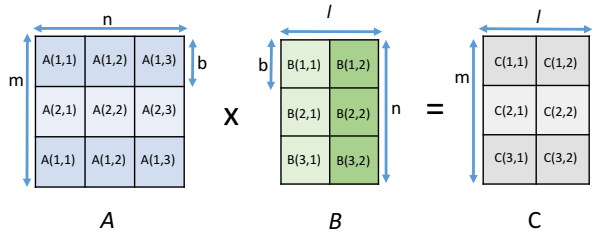
The commonality in these and other similar results is that they divide the input matrices into row and column blocks, where each worker multiplies a row block (or some combination of row blocks) of $\mathbf{A}$ and a column block (or some combination of column blocks) of $\mathbf{B}$. These methods provide straggler resiliency but are not cost-efficient as they require asymptotically more communication than blocked partitioning of data, as discussed in detail in the next section. Another disadvantage of such coding-based methods is that there are separate encoding and decoding phases that require additional communication and potentially large computational burden at the master node, which may make the algorithm infeasible in some distributed computing environments.

3.2 Preliminaries

There are two common schemes for distributed multiplication of two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times l}$, as illustrated in Figures 3.2a and 3.2b. We refer to these schemes as *naive* and *blocked* matrix multiplication, respectively. Detailed steps for these schemes are provided in Algorithms 3 and 4, respectively, for the serverless setting. During naive matrix multiplication, each worker receives and multiplies an $a \times n$ row-block of $\mathbf{A}$ and $n \times a$ column-block of $\mathbf{B}$ to compute an



(a) Distributed naive matrix multiplication, where each worker multiplies a row-block of $\mathbf{A}$ of size $a \times n$ and a column block of $\mathbf{B}$ of size $n \times a$ to get an $a \times a$ block of $\mathbf{C}$.



(b) Distributed blocked matrix multiplication, where each worker multiplies a sub-block of $\mathbf{A}$ of size $b \times b$ and a sub-block of $\mathbf{B}$ of size $b \times b$.

Figure 3.2: An illustration of two algorithms for distributed matrix multiplication.

$a \times a$ block of $\mathbf{C}$. Blocked matrix multiplication consists of two phases. During the computation phase, each worker gets two $b \times b$ blocks, one each from $\mathbf{A}$ and $\mathbf{B}$, which are then multiplied by the workers. In the reduction phase, to compute a $b \times b$ block of $\mathbf{C}$, one worker gathers results of all the n/b workers from the cloud storage corresponding to one row-block of $\mathbf{A}$ and one column-block of $\mathbf{B}$ and adds them. For example, in Figure 3.2b, to get $\mathbf{C}(1,1)$, results from 3 workers who compute $\mathbf{A}(1,1) \times \mathbf{B}(1,1)$, $\mathbf{A}(1,2) \times \mathbf{B}(2,1)$ and $\mathbf{A}(1,3) \times \mathbf{B}(3,1)$ are added.

It is accepted in High Performance Computing (HPC) that blocked partitioning of input matrices takes less time than naive matrix multiplication [113], [114], [130]. For example, in [113], the authors propose 2.5D matrix multiplication, an optimal communication avoiding algorithm for matrix multiplication in HPC/server-based computing, that divides input matrices into blocks and stores redundant copies of them across processors to reduce bandwidth and latency costs. However, perhaps due to lack of a proper analysis for cloud-based distributed computing, existing algorithms for straggler mitigation in the cloud do naive matrix multiplication [26]–[28]. Next, we bridge the gap between cost analysis and straggler mitigation for distributed computation in the serverless setting.

3.3 Cost Analysis: Naive and Blocked multiplication

There are communication and computation costs associated with any distributed algorithm. Communication costs themselves are of two types: latency and bandwidth. For example, sending n bits requires packing them into contiguous memory and transmitting them as a message. The latency

Algorithm 3: Distributed naive matrix multiplication

Input : Matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times l}$
Result: $\mathbf{C} = \mathbf{A} \times \mathbf{B}$
Initialization: Divide $\mathbf{A}$ into submatrices of size $a \times n$ (row-wise division) and $\mathbf{B}$ into submatrices of size $n \times a$ (column-wise division)
for *i=1 to m/a* **do**
  for *j=1 to l/a* **do**
    1. Worker W_{ij} receives i-th chunk of $\mathbf{A}$, say $\mathbf{A}(i,:)$, and j-th chunk of $\mathbf{B}$, say $\mathbf{B}(:,j)$
    2. W_{ij} computes the $a \times a$ chunk of $\mathbf{C}$, that is, $\mathbf{C}(i,j) = \mathbf{A}(i,:) \times \mathbf{B}(:,j)$
    3. W_{ij} writes $\mathbf{C}(i,j)$ back to the cloud storage
  end
end

cost α is the fixed overhead time spent in packing and transmitting a message over the network. Thus, to send Q messages, the total latency cost is αQ. Similarly, to transmit K bits, a bandwidth cost proportional to K, given by βK, is associated. Letting γ denote the time to perform one floating point operation (FLOP), the total computing cost is γF, where F is the total number of FLOPs at the node. Hence, the total time pertaining to one node that sends M messages, K bits and performs F FLOPs is

$$T_{\text{worker}} = \alpha Q + \beta K + \gamma F,$$

where $\alpha \gg \beta \gg \gamma$. The (α, β, γ) model defined above has been well-studied and is used extensively in the HPC literature [113], [129], [130], [132], [133]. It is ideally suited for serverless computing, where network topology does not affect the latency costs as each worker reads/writes directly from/to the cloud storage and no multicast gains are possible.

However, our analysis for costs incurred during distributed matrix multiplication differs from previous works in three principle ways. 1) Workers in serverless architecture cannot communicate amongst themselves, and hence, our algorithm for blocked multiplication is very different from optimal communication avoiding algorithm for HPC that involves message passing between workers [113]. 2) The number of workers in HPC analyses is generally fixed, whereas the number of workers in the serverless setting is quite flexible, easily scaling into the thousands, and the limiting factor is memory/bandwidth available at each node. 3) Computation on the inexpensive cloud is more motivated by savings in expenditure than the time required to run the algorithm. We define our cost function below.

If there are W workers, each doing an equal amount of work, the total amount of money spent in running the distributed algorithm on the cloud is proportional to

$$C_{\text{total}} = W \times T_{\text{worker}} = W(\alpha Q + \beta K + \gamma F). \tag{3.1}$$

Eq. (3.1) does not take into account the straggling costs as they increase the total cost by a constant factor (by re-running the jobs that are straggling) and hence does not affect our asymptotic analysis.

Inexpensive nodes in serverless computing are generally constrained by the amount of memory or communication bandwidth available. For example, AWS Lambda nodes have a maximum

Algorithm 4: Distributed blocked matrix multiplication

Input : Matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times l}$
Result: $\mathbf{C} = \mathbf{A} \times \mathbf{B}$
Initialization: Divide $\mathbf{A}$ into $m/b \times n/b$ matrix and $\mathbf{B}$ into $n/b \times l/b$ matrix of $b \times b$ blocks where b is the block-size
// Computation phase:
for *i = 1 to m/b* **do**
  for *j = 1 to l/b* **do**
    for *k = 1 to n/b* **do**
      1. Worker W_{ijk} gets (i,k)-th block of $\mathbf{A}$, say $\mathbf{A}(i,k)$, and (k,j)-th block of $\mathbf{B}$, say $\mathbf{B}(k,j)$, from the cloud storage
      2. W_{ijk} then computes the $b \times b$ product $\hat{\mathbf{C}}_{ijk} = \mathbf{A}(i,k) \times \mathbf{B}(k,j)$
      3. Worker writes the result W_{ijk} back to the cloud storage
    end
  end
end
// Reduction phase:
for *i = 1 to m/b* **do**
  for *j = 1 to l/b* **do**
    Spin a new worker, say W_{ij}, that stores an all-zero $b \times b$ sub-block C_{ij}
    for *k = 1 to n/b* **do**
      1. W_{ij} extracts the output $\hat{\mathbf{C}}_{ijk}$ written by W_{ijk} from cloud storage
      2. W_{ij} does $\mathbf{C}_{ij} = \mathbf{C}_{ij} + \hat{\mathbf{C}}_{ijk}$
    end
    W_{ij} writes $\mathbf{C}_{ij}$ back to the cloud storage
  end
end

allocated memory of 3008 MB[1], a fraction of the memory available in today's smartphones. Let the memory available at each node be limited to M. That is, the communication bandwidth available at each worker is limited to M, and this is the main bottleneck of the distributed system. We would like to multiply two large matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times l}$ in parallel, and let $M = O(n^{\delta})$. Note that if $\delta \geq 2$, one of the following will happen:

- $m = O(n)$ and $l = O(n)$, and the input matrices can fit into one worker's memory and parallelism is not required.
- Either $m = \omega(n)$ or $l = \omega(n)$ or both, and block-size for blocked matrix multiplication is n. The two schemes, naive and blocked multiplication, would exactly be the same in this case.

Thus, for all practical cases in consideration, $\delta < 2$.

Theorem 3. *For the cost model defined in Eq.* (3.1)*, communication (i.e., latency and bandwidth) costs for blocked multiplication outperform naive multiplication by a factor of* $O(n^{1-\delta/2})$*, where the individual costs are listed in Table 3.1.*

[1] AWS Lambda limits are available at (may change over time) https://docs.aws.amazon.com/lambda/latest/dg/limits.html

Table 3.1: Costs comparison for naive and blocked matrix multiplication in the serverless setting, where $\delta < 2$.

Cost type	Naive multiply	Blocked Multiply	Ratio: naive/blocked
Latency	$O(mln^{2(1-\delta)})$	$O(mln^{1-3\delta/2})$	$O(n^{1-\delta/2})$
Bandwidth	$O(mln^{2-\delta})$	$O(mln^{1-\delta/2})$	$O(n^{1-\delta/2})$
Computation	$O(mln)$	$O(mln)$	1

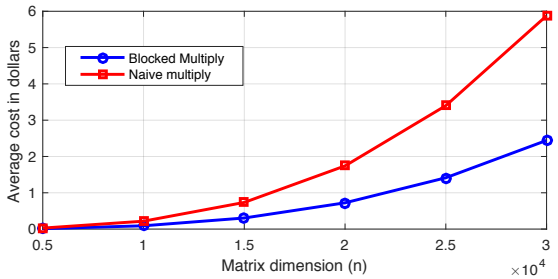


Figure 3.3: Comparison of AWS Lambda costs for multiplying two $n \times n$ matrices, where each worker is limited by 3008 MB of memory and price per running worker per 100 milliseconds is $\$0.000004897$.

Proof. See Section 3.6.1. □

The rightmost column in Table 3.1 lists the ratio of communication costs for naive and blocked matrix multiplication. We note that the latter significantly outperforms the former, with communication costs being asymptotically worse for naive multiplication. An intuition behind why this happens is that each worker in distributed blocked multiplication does more work than in distributed naive multiplication for the same amount of received data. For example, to multiply two square matrices of dimension n, where memory at each worker limited by $M = 2n$, $a = 1$ for naive multiplication and $b = \sqrt{n}$ for blocked multiplication. We note that the amount of work done by each worker in naive and blocked multiplication is $O(n)$ and $O(n^{3/2})$, respectively. Since the total amount of work is constant and equal to $O(n^3)$, blocked matrix multiplication ends up communicating less during the overall execution of the algorithm as it requires fewer workers. Note that naive multiplication takes less time to complete as each worker does asymptotically less work, however, the number of workers required is asymptotically more, which is not an efficient utilization of resources and increases the expenditure significantly.

Remark 4. For clarity of exposition, we partition the input matrices into square-blocks of dimension b. However, optimal block dimensions for rectangular-block partitions can be found by minimizing (3.1) while incorporating the memory constraints of the distributed system. This is a convex problem that can be converted to a geometric program [134]. We note that the optimal block dimension found in this way is nearly square.

Figure 3.3 supports the above analysis where we plot the cost in dollars of multiplying two square matrices in AWS Lambda, where each node's memory is limited by 3008 MB and price

per worker per 100 millisecond is $\$0.000004897$. However, as discussed earlier, existing schemes for straggler-resiliency in distributed matrix multiplication consider naive multiplication which is impractical from a user's point of view. In the next section, we propose OverSketch, a scheme to mitigate the detrimental effects of stragglers for blocked matrix multiplication.

3.4 OverSketch: Straggler-resilient Blocked Matrix Multiplication using Sketching

Many of the recent advances in algorithms for numerical linear algebra have come from the technique of linear sketching, in which a given matrix is compressed by multiplying it with a random matrix of appropriate dimension [40]–[43]. Sketching accelerates computation by eliminating redundancy in the matrix structure through dimension reduction. However, the coding-based approaches described in Section 3.1.2 have shown that redundancy can be *good* for combating stragglers if judiciously introduced into the computation. With these competing points of view in mind, our algorithm OverSketch works by "oversketching" the matrices to be multiplied by reducing dimensionality not to the minimum required for sketching accuracy, but rather to a slightly higher amount which simultaneously ensures both the accuracy guarantees and speedups of sketching *and* the straggler resilience afforded by the redundancy which was not eliminated in the sketch. OverSketch further reduces asymptotic costs by adopting the idea of block partitioning from HPC, suitably adapted for a serverless architecture.

Next, we propose a sketching scheme for OverSketch and describe the process of straggler mitigation in detail.

3.4.1 OverSketch: The Algorithm

During blocked matrix multiplication, the (i, j)-th block of $\mathbf{C}$ is computed by assimilating results from d/b workers who compute the product $\tilde{\mathbf{A}}(i, k) \times \tilde{\mathbf{B}}(k, j)$, for $k = 1, \cdots, d/b$. Thus, the computation $\mathbf{C}(i, j)$ can be viewed as the product of the row sub-block $\tilde{\mathbf{A}}(i, :) \in \mathbb{R}^{b \times d}$ of $\tilde{\mathbf{A}}$ and the column sub-block $\tilde{\mathbf{B}}(:, j) \in \mathbb{R}^{d \times b}$ of $\tilde{\mathbf{B}}$. An illustration is shown in Figure 3.4. Assuming d is large enough to guarantee the required accuracy in $\mathbf{C}$, we increase the sketch dimension from d to $z = d + eb$, where e is the worst case number of stragglers in $N = d/b$ workers. For the example in Figure 3.4, $e = 1$. To get a better insight on e, we observe in our simulations for cloud systems like AWS lambda and EC2 that the number of stragglers is $< 5\%$ for most runs. Thus, if $N = d/b = 40$, i.e. 40 workers compute one block of $\mathbf{C}$, then $e \approx 2$ is sufficient to get similar accuracy for matrix multiplication. We describe OverSketch in detail in Algorithm 5. Next, we describe how to compute the sketched matrices $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$.

Many sketching techniques have been proposed recently for approximate matrix computations. For example, to sketch a $m \times n$ matrix $\mathbf{A}$ with sketch dimension d, Gaussian projection takes $O(mnd)$ time, Subsampled Randomized Hadamard Transform (SRHT) takes $O(mn \log n)$ time, and Count-sketch takes $O(nnz(A))$ time, where $nnz(\cdot)$ is the number of non-zero entries [41], [135]–[137].

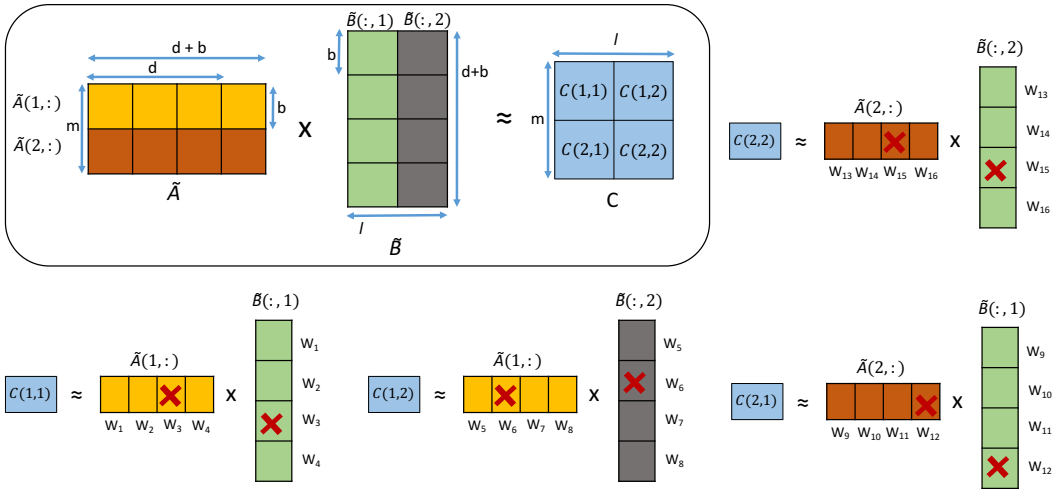


Figure 3.4: An illustration of multiplication of $m \times z$ matrix $\tilde{\mathbf{A}}$ and $z \times l$ matrix $\tilde{\mathbf{B}}$, where $z = d + b$ assures resiliency against one straggler per block of $\mathbf{C}$, and d is chosen by the user to guarantee a desired accuracy. Here, $m = l = 2b$, $d = 3b$, where b is the block-size for blocked matrix multiplication. This scheme ensures one worker can be ignored while calculating each block of $\mathbf{C}$.

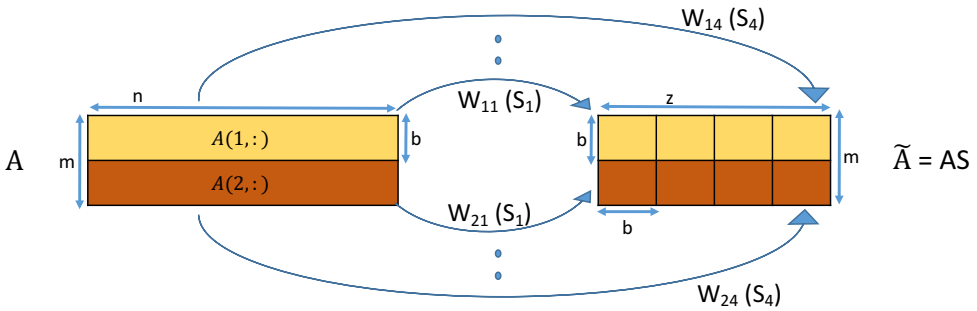


Figure 3.5: An illustration of sketching $\mathbf{A} \in \mathbb{R}^{m \times n}$ in parallel using the sketch matrix in Eq. (3.3) with sketch dimension $z = (N + e)b$. Worker W_{ij} receives the row-block $\mathbf{A}(i, :)$ of $\mathbf{A}$ and the Count-sketch $\mathbf{S}_j$ to compute the (i, j)-th block of $\tilde{\mathbf{A}}$. Sketching requires a total of mz/b^2 workers. Here, $z = 4b$, $N = 3$ and $e = 1$, and $\mathbf{A}$ is divided into 2 row-blocks, that is, $m = 2b$. Total number of workers required for distributed sketching is 8.

Algorithm 5: OverSketch: Distributed blocked matrix multiplication for the Cloud

Input : Matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times l}$, sketch dimension z, straggler tolerance e
Result: $\mathbf{C} \approx \mathbf{A} \times \mathbf{B}$

1. **Sketching**: Use Algorithm 7 to obtain $\tilde{\mathbf{A}} = \mathbf{AS}$ and $\tilde{\mathbf{B}} = \mathbf{S}^T\mathbf{B}$ distributedly
2. **Block partitioning**: Divide $\tilde{\mathbf{A}}$ into $m/b \times z/b$ matrix and $\mathbf{B}$ into $z/b \times l/b$ matrix of $b \times b$ blocks where b is the block-size
3. **Computation phase**: Use the computation phase from Algorithm 4 to multiply $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$. This step invokes mlz/b^3 workers, where z/b workers are used per block of $\mathbf{C}$
4. **Termination**: Stop computation when any d/b workers return their results for each of the ml/b^2 blocks of $\mathbf{C}$, where $d = z - eb$
5. **Reduction phase**: Invoke ml/b^2 workers for reduction as described in Algorithm 4 on available results

Algorithm 6: Calculating Count-sketch of a matrix $\mathbf{A}$

Input : Matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and sketch dimension b
Result: $\tilde{\mathbf{A}} \in \mathbb{R}^{m \times b}$, a random Count-sketch of $\mathbf{A}$

1. Multiply each column of $\mathbf{A}$ by -1 with probability 0.5
2. Map each column of resultant $\mathbf{A}$ to an integer in $[1, b]$ uniformly randomly
3. Add columns in $\mathbf{A}$ that are mapped to the same integer. The resultant matrix is a Count-sketch of $\mathbf{A}$

Count sketch, first exploited in data streaming literature, has been widely applied to expedite large-scale matrix computations [135], [137], [138]. It is one of the most popular sketching techniques as it requires linear time to compute the matrix sketch with similar approximation guarantees for matrix multiplication. To compute the Count-sketch of $\mathbf{A} \in \mathbb{R}^{m \times n}$ of sketch dimension b, each column in $\mathbf{A}$ is multiplied by -1 with probability 0.5 and then mapped to an integer sampled uniformly from $\{1, 2, \cdots, b\}$. Then, to compute the sketch $\tilde{\mathbf{A}}_c = \mathbf{AS}_c$, columns with the same mapped value are summed (see Algorithm 6 for details). An example of Count-sketch matrix with $n = 9$ and $b = 3$ is

$$\mathbf{S}_c^T = \begin{bmatrix} 0 & 0 & 0 & 1 & -1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & -1 \end{bmatrix}. \tag{3.2}$$

Here, $\mathbf{A}$ has 9 columns, and columns $4, 5$ and 7 were mapped to 1, columns $1, 2$ and 6 were mapped to 2, and columns $3, 8$ and 9 were mapped to 3. Thus, the Count-sketch $\tilde{\mathbf{A}}_c$ would have only 3 columns, which are obtained by summing the columns of $\mathbf{A}$ with the same mapped value (after possibly multiplying with -1). The sparse structure of $\mathbf{S}_c$ ensures that the computation of sketch takes $O(nnz(A))$ time. However, a drawback of the desirable sparse structure of Count-sketch is that it cannot be directly employed for straggler mitigation in blocked matrix multiplication as it would imply complete loss of information from a subset of columns of $\mathbf{A}$. For the example in (3.2), suppose the worker processing column 3 of $\tilde{\mathbf{A}}_c$ be straggling. Ignoring this worker would imply

that columns $3, 8$ and 9 of $\mathbf{A}$ were not considered in the computation. This will generally lead to poor accuracy for sketched matrix multiplication.

To facilitate straggler mitigation for blocked matrix multiplication, we propose a new sketch matrix $\mathbf{S}$, inspired by Count-sketch, and define it as

$$\mathbf{S} = \frac{1}{\sqrt{N}}(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N+e}), \tag{3.3}$$

where $N = d/b$, e is the expected number of stragglers per block of $\mathbf{C}$ and $\mathbf{S}_i \in \mathbb{R}^{n\times b}$, for $i = 1, 2, \cdots, (N + e)$, is a Count-sketch matrix with dimension b. Thus, the total sketch-dimension for the sketch matrix in (3.3) is $z = (N + e)b = d + eb$. Computation of this sketch takes $O(nnz(\mathbf{A})(N + e))$ time in total and can be implemented in a distributed fashion trivially, where $(N + e)$ is the number of workers per block of $\mathbf{C}$. An illustration of distributed sketching of $\mathbf{A}$ in serverless systems is described in Figure 3.5. For detailed steps, see Algorithm 7. A few remarks regarding OverSketch based distributed matrix multiplication are in order.

- **Graceful degradation**: Coding-based straggler mitigation (see Figure 3.1 for example) cannot tolerate more stragglers than provisioned. An advantage of using sketching schemes for computation is that more stragglers can be tolerated than initially provisioned at the cost of accuracy of the result, thus exhibiting 'graceful degradation'.

- **Memory constrained distributed sketching**: It is assumed in Algorithm 7 that each worker can store an entire row-block of $\mathbf{A}$ in memory to calculate its Count-sketch. That might not always be the case, especially in low-memory serverless nodes. However, since Count-sketch is a streaming based algorithm, workers can calculate the sketch by further partitioning the $b \times n$ row-block into $b \times b$ square blocks and copying only one block at a time (see [71] for details).

- **Straggler-resilient sketching**: We note that the stragglers can also be ignored during distributed sketching (Algorithm 7). More specifically, the blocks ignored during the sketching phase can be marked as faults/stragglers during computation phase in Algorithm 5.

- **Limitations of "Over-sampling"**: Schemes like leverage-score based sampling are also used in literature to compute approximate product of $\mathbf{A}$ and $\mathbf{B}$ [40], [139]. Such schemes are as efficient as Count-sketch but are not suitable for straggler-resilient blocked multiplication. For example, if a worker with a block of $\tilde{\mathbf{A}}$ straggles, where $\tilde{\mathbf{A}}$ is obtained by "over"-sampling the columns of $\mathbf{A}$ according to leverage scores, results from all other workers that are working on that column-block of $\tilde{\mathbf{A}}$ is wasted as part of the column-block is unavailable. Thus, oversampling can require huge redundancy even for a small number of stragglers.

Next, we prove statistical guarantees on the accuracy of our sketching based matrix multiplication algorithm.

Algorithm 7: "Over" sketching $\mathbf{A}$ in parallel using the sketch $\mathbf{S}$ in (3.3) to compute $\tilde{\mathbf{A}} = \mathbf{AS}$

Input : Matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and "Over" sketch dimension z
Result: $\tilde{\mathbf{A}} = \mathbf{A} \times \mathbf{S}$
Initialization: Divide $\mathbf{A}$ into row-blocks of size $b \times n$
for *i=1 to m/b* **do**
  for *j=1 to z/b* **do**
    1. Worker W_{ij} receives i-th row-block of $\mathbf{A}$, say $\mathbf{A}(i,:)$
    2. W_{ij} uses Algorithm 6 to compute the (i,j)-th $b \times b$ block of $\tilde{\mathbf{A}}$ using Count-sketch $\mathbf{S}_j$, that is, $\tilde{\mathbf{A}}(i,j) = \mathbf{A}(i,:) \times \mathbf{S}_j$
    3. W_{ij} writes $\tilde{\mathbf{A}}(i,j)$ back to the cloud storage
  end
end

3.4.2 OverSketch: Approximation guarantees

Definition 2. We say that an approximate matrix multiplication of two matrices $\mathbf{A}$ and $\mathbf{B}$ using sketch $\mathbf{S}$, given by $\mathbf{ASS}^T\mathbf{B}$, is (ϵ, θ) accurate if, with probability at least $(1-\theta)$, it satisfies

$$||\mathbf{AB} - \mathbf{ASS}^T\mathbf{B}||_F^2 \leq \epsilon ||\mathbf{A}||_F^2 ||\mathbf{B}||_F^2.$$

Now, for blocked matrix multiplication using OverSketch and as illustrated in Figure 3.4, the following holds

Theorem 4. *Computing* $(\mathbf{AS}) \times (\mathbf{S}^T\mathbf{B})$ *using sketch* $\mathbf{S} \in \mathbb{R}^{n \times z}$ *in* (3.3) *and* $d = \frac{2}{\epsilon\theta}$*, while ignoring* e *stragglers among any* $\frac{z}{b}$ *workers, is* (ϵ, θ) *accurate.*

Proof. See Appendix 3.6.2.

For certain applications, the guarantee in theorem 4 may be too crude as the product of $||\mathbf{A}||_F^2$ and $||\mathbf{B}||_F^2$ in the RHS can get big for large matrices $\mathbf{A}$ and $\mathbf{B}$. We can obtain a stronger result than in theorem 4 when $\min(rank(\mathbf{A}), rank(\mathbf{B})) \ll n$, for example, when $\mathbf{A}$ is a fat matrix, or $\mathbf{B}$ is a tall matrix. Without loss of generality, say $\min(rank(\mathbf{A}), rank(\mathbf{B})) = rank(\mathbf{A}) = r$. Thus, $||\mathbf{A}||_2 \leq ||\mathbf{A}||_F \leq \sqrt{r}||\mathbf{A}||_2$, where $||\cdot||_2$ denotes the spectral norm. Hence, with probability at least $(1-\theta)$

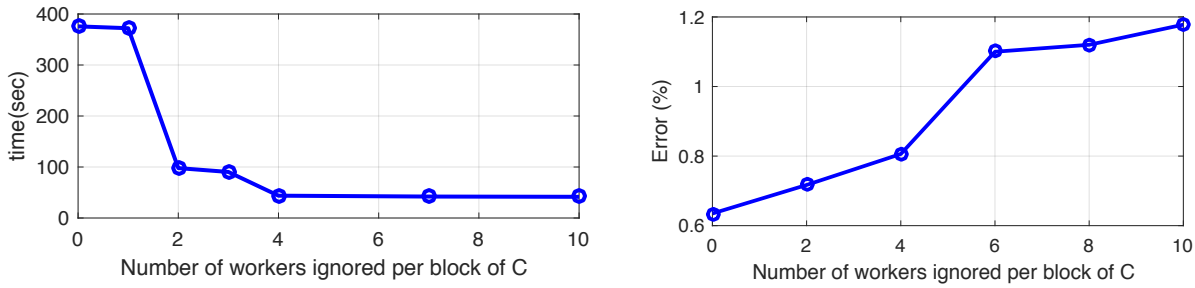
$$||\mathbf{ASS}^T\mathbf{B} - \mathbf{AB}||_F^2 \leq \epsilon r ||\mathbf{A}||_2^2 ||\mathbf{B}||_F^2.$$

Now, if we increase the sketch dimension by a factor of r to $z = r(d + eb) = O(\frac{r}{\epsilon\theta})$, we get

$$||\mathbf{ASS}^T\mathbf{B} - \mathbf{AB}||_F^2 \leq \epsilon ||\mathbf{A}||_2^2 ||\mathbf{B}||_F^2 \tag{3.4}$$

with probability $(1-\theta)$, which is a better approximation for the product $\mathbf{ASS}^T\mathbf{B}$.

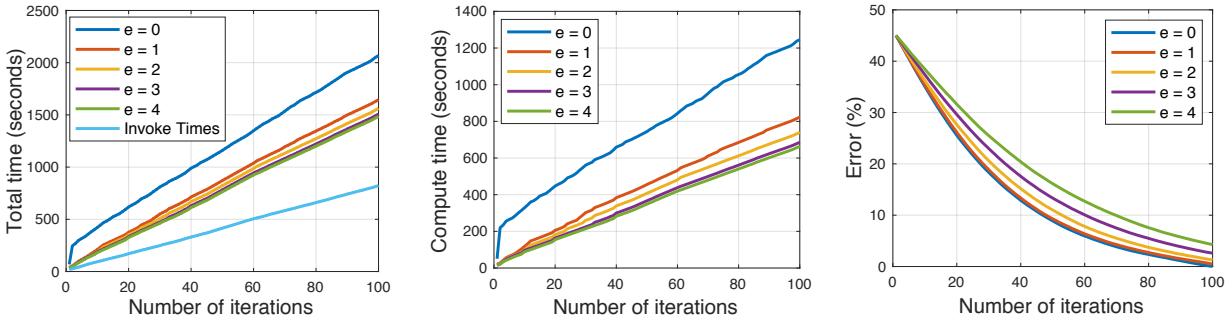
During the reduction phase, we use ml/b^2 workers, which is much less than the number of workers used during the computation phase, that is, mlz/b^3. In our experiments, we observe that the possibility of stragglers reduces significantly if fewer workers are used. This is especially true for



(a) Time statistics for OverSketch on AWS Lambda for the straggler profile in Figure 1.2

(b) Frobenius norm error for sketched matrix product

Figure 3.6: Time and approximation error for OverSketch with 3000 workers when e, the number of workers ignored per block of $\mathbf{C}$, is varied from 0 to 10.



(a) Plot of total time (i.e., invocation time plus computation time) versus number of iterations.

(b) Plot of computation time versus number of iterations. When $e = 1$, algorithm takes 7 minutes less compared to when $e = 0$.

(c) Plot of percentage error versus iterations. Ignoring one worker per block of $\mathbf{C}$ has negligible affect on the convergence.

Figure 3.7: Time statistics and optimality gap on AWS Lambda while solving the LP in (3.5) using interior point methods, where e is the number of workers ignored per block of $\mathbf{C}$.

the reduction phase, as healthy running workers from the computation phase are reused, reducing the chances of stragglers. However, in the unfortunate event that stragglers are observed during reduction, speculative execution can be used, i.e. detecting and restarting the slow job. Another simple solution is to use existing coding techniques as described in Figure 3.1, that is, by adding one parity row-block to $\tilde{\mathbf{A}}$ and one parity row column to $\tilde{\mathbf{B}}$ before multiplying them, which can tolerate 3 stragglers in the worst case. However, this would require a decoding step to compensate for the missing stragglers.

3.5 Experimental Results

3.5.1 Blocked Matrix Multiplication on AWS Lambda

We implement the straggler-resilient blocked matrix multiplication described above in the serverless computing platform *Pywren* [1], [3][2], on the AWS Lambda cloud system to compute an approximate $\mathbf{C} = \mathbf{AS} \times \mathbf{S}^T\mathbf{B}$ with $b = 2048, m = l = 10b, n = 60b$ and $\mathbf{S}$ as defined in (3.3) with sketch dimension $z = 30b$. Throughout this experiment, we take $\mathbf{A}$ and $\mathbf{B}$ to be constant matrices where the entries of $\mathbf{A}$ are given by $\mathbf{A}(x, y) = x + y$ for all $x \in [1, m]$ and $y \in [1, n]$ and $\mathbf{B} = \mathbf{A}^T$. Thus, to compute (i, j)-th $b \times b$ block of $\mathbf{C}$, 30 nodes compute the product of $\tilde{\mathbf{A}}(i, :)$ and $\tilde{\mathbf{B}}(:, j)$, where $\tilde{\mathbf{A}} = \mathbf{AS}$ and $\tilde{\mathbf{B}} = \mathbf{S}^T\mathbf{B}$. While collecting results, we ignore e workers for each block of $\mathbf{C}$, where e is varied from 0 to 10.

The time statistics are plotted in Figure 3.6a. The corresponding worker job times are shown in Figure 1.2, where the median job time is around 42 seconds, and some stragglers return their results around 100 seconds and some others take up to 375 seconds. We note that the compute time for matrix multiplication reduces by a factor of 9 if we ignore at most 4 workers per 30 workers that compute a block of $\mathbf{C}$. In figure 3.6b, for same $\mathbf{A}$ and $\mathbf{B}$, we plot average error in matrix multiplication by generating ten instances of sketches and averaging the error in Frobenius norm, $\frac{||\mathbf{AB}-\mathbf{ASS}^T\mathbf{B}||_F}{||\mathbf{AB}||_F}$, across instances. We see that the average error is only 0.8% when 4 workers are ignored.

3.5.2 Solving Optimization Problems with Sketched Matrix multiplication

Matrix multiplication is the bottleneck of many optimization problems. Thus, sketching has been applied to solve several fairly common optimization problems using second-order methods, like linear programs, maximum likelihood estimation, generalized linear models like least squares and logistic regression, semi-definite programs, support vector machines, Kernel ridge regression, etc., with essentially same convergence guarantees as exact matrix multiplication [42], [43]. As an instance, we solve the following linear program (LP) using interior point methods on AWS Lambda

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize}}\ \mathbf{c}^T\mathbf{x} \\ &\text{subject to } \mathbf{Ax} \leq \mathbf{b}, \end{aligned} \tag{3.5}$$

where $\mathbf{x} \in \mathbb{R}^{m\times 1}, \mathbf{c} \in \mathbb{R}^{m\times 1}, \mathbf{b} \in \mathbb{R}^{n\times 1}$ and $\mathbf{A} \in \mathbb{R}^{n\times m}$ is the constraint matrix with $n > m$. To solve (3.5) using the logarithmic barrier method, we solve the following sequence of problems using Newton's method

$$\min_{\mathbf{x}\in\mathbb{R}^m} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^m} \left(\tau\mathbf{c}^T\mathbf{x} - \sum_{i=1}^{n} \log(b_i - \mathbf{a}_i\mathbf{x}) \right), \tag{3.6}$$

[2]A working implementation of OverSketch is available at https://github.com/vvipgupta/OverSketch

where $\mathbf{a}_i$ is the i-th row of $\mathbf{A}$, τ is increased geometrically as $\tau = 2\tau$ after every 10 iterations and the total number of iterations is 100. The update in the t-th iteration is given by

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta(\nabla^2 f(\mathbf{x}_t))^{-1}\nabla f(\mathbf{x}_t), \tag{3.7}$$

where $\mathbf{x}_t$ is the estimate of the solution in the t-th iteration and η is the appropriate step-size. The gradient and Hessian for the objective in (3.6) are given by

$$\nabla f(\mathbf{x}) = \tau\mathbf{c} + \sum_{i=1}^{n} \frac{\mathbf{a}_i^T}{b_i - \mathbf{a}_i^T\mathbf{x}} \text{ and} \tag{3.8}$$

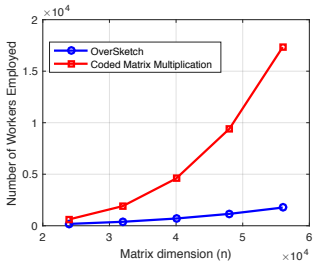
$$\nabla^2 f(\mathbf{x}) = \mathbf{A}^T \text{diag}\frac{1}{(bi - \mathbf{a}_i\mathbf{x})^2}\mathbf{A}, \tag{3.9}$$

respectively. The square root of the Hessian is given by $\nabla^2 f(\mathbf{x})^{1/2} = \text{diag}\frac{1}{|bi-\mathbf{a}_i\mathbf{x}|}\mathbf{A}$. The computation of Hessian requires $O(nm^2)$ time and is the bottleneck in each iteration. Thus, we use our distributed and sketching-based blocked matrix multiplication scheme to mitigate stragglers while evaluating the Hessian approximately, i.e. $\nabla^2 f(\mathbf{x}) \approx (\mathbf{S}\nabla^2 f(\mathbf{x})^{1/2})^T \times (\mathbf{S}\nabla^2 f(\mathbf{x})^{1/2})$, on AWS Lambda, where $\mathbf{S}$ is defined in (3.3).

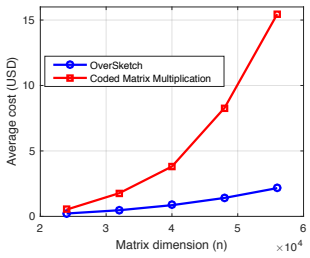
We take the block size, b, to be 1000, the dimensions of $\mathbf{A}$ to be $n = 40b$ and $m = 5b$ and the sketch dimension to be $z = 20b$. We use a total of 500 workers in each iteration. Thus, to compute each $b \times b$ block of $\mathbf{C}$, 20 workers are assigned to compute matrix multiplication on two $b \times b$ blocks. We depict the time and error versus iterations in figure 3.7. We plot our results for different values of e, where e is the number of workers ignored per block of $\mathbf{C}$. In our simulations, each iteration includes around 9 seconds of invocation time to launch AWS Lambda workers and assign tasks. In figure 3.7a, we plot the total time that includes the invocation time and computation time versus iterations. In 3.7b, we exclude the invocation time and plot just the compute time in each iteration and observe 34% savings in solving (3.5) when $e = 1$, whereas the effect on the error with respect to the optimal solution is insignificant (as shown in figure 3.7c).

3.5.3 Comparison with Existing Straggler Mitigation Schemes

In this section, we compare OverSketch with an existing coding-theory based straggler mitigation scheme described in [26]. An illustration for [26] is shown in Figure 3.1. We multiply two square matrices $\mathbf{A}$ and $\mathbf{B}$ of dimension n on AWS Lambda using the two schemes, where $\mathbf{A}(x, y) = x + y$ and $\mathbf{B}(x, y) = x \times y$ for all $x, y \in [1, n]$. We limit the bandwidth of each worker by 400 MB (i.e. around 48 million entries, where each entry takes 8 bytes) for a fair comparison. Thus, we have $3b^2 = 48 \times 10^6$, or $b = 4000$ for OverSketch and $2an + a^2 = 48 \times 10^6$ for [26], where a is the size of the row-block of $\mathbf{A}$ (and column-block of $\mathbf{B}$). We vary the matrix dimension n from $6b = 24000$ to $14b = 56000$. For OverSketch, we take the sketch dimension z to be $n/2 + b$, and take $e = 1$, i.e., ignore one straggler per block of $\mathbf{C}$. For straggler mitigation in [26], we add one parity row in $\mathbf{A}$ and one parity column in $\mathbf{B}$. In Figures 3.8a and 3.8b, we compare the workers required and average cost in dollars, respectively, for the two schemes. We note that OverSketch requires



(a) Number of workers required for OverSketch and [26]



(b) Cost for distributed matrix multiplication for OverSketch and [26]

Figure 3.8: Comparison of OverSketch with coded theory based scheme in [26] on AWS Lambda. OverSketch requires asymptotically less workers which translates to significant savings in cost.

asymptotically fewer workers, and it translates to the cost for doing matrix multiplication. This is because the running time at each worker is heavily dependent on communication, which is the same for both the schemes. For $n = 20000$, the average error in Frobenius norm for OverSketch is less than 2%, and decreases as n is increased.

The scheme in [26] requires an additional decoding phase, and assume the existence of a powerful master that can store the entire product $\mathbf{C}$ in memory and decode for the missing blocks using the redundant chunks. This is also true for the other schemes in [27]–[29]. Moreover, these schemes would fail when the number of stragglers is more than the provisioned redundancy while OverSketch has a 'graceful degradation' as one can get away by ignoring more workers than provisioned at the cost of accuracy of the result.

3.6 Proofs

3.6.1 Proof of Theorem 3

To compare naive and blocked multiplication, we first observe that the computation cost in (3.1), that is $W \times F$, is the same for both naive and blocked multiplication and is equal to $O(mnl)$, which is the total amount of work done during matrix-matrix multiplication[3]. Let W_1 be the number of workers required for naive matrix multiplication. Then, $W_1 = ml/a^2$, as each worker is sent one row-block of $\mathbf{A}$ from m/a choices, and one column-block of $\mathbf{B}$ from l/a choices. Each worker receives $2an$ entries and writes back a^2 entries. Hence, the total communication incurred during the algorithm is $W_1 \times (2an + a^2) = (2nml/a + ml)$. Also, since each worker can only receive M entries, we have $M = 2an$, thus $a = M/2n$. Hence, the total bandwidth cost for naive multiplication is $\beta \times (4n^2ml/M + ml) = O(n^{2-\delta}ml)$. Also, the total number of messages sent during the process is W_1, and hence the total latency cost is $O(mln^{2(1-\delta)})$.

[3]The computation cost for blocked matrix multiplication can be further improved by using Strassen type methods that take $O(b^{2.38})$ to multiply two square sub-blocks of dimension $b \times b$, but we do not consider that advantage in this chapter for clarity of exposition and to emphasize on savings just due to communication.

During the computation phase for blocked multiplication, $W_{2,comp} = (n/b) \times ml/b^2$, as computation of one $b \times b$ block of $\mathbf{C} \in \mathbb{R}^{m \times l}$ requires n/b workers, and there are a total of ml/b^2 such blocks. Again, each worker receives two $b \times b$ blocks, one from each $\mathbf{A}$ and $\mathbf{B}$, and writes back a $b \times b$ block, where b satisfies $M = 2b^2$. Thus, the total bandwidth cost incurred during the computation phase is $\beta W_{2,comp} \times 3b^2 = 3\beta nml/b = O(mln^{1-\delta/2})$. The total number of messages received by the workers is $W_{2,comp}$, and, hence, the latency cost is $\alpha nml/b^3 = O(mln^{1-3\delta/2})$. During the reduction phase, the number of workers required is $W_{2,red} = ml/b^2$, and each worker receives n/b blocks of size $b \times b$ to compute one block of $\mathbf{C}$. Thus, for the reduction phase, the communication is $W_{2,red} \times b^2 \times (n/b) = nml/b = O(mln^{1-\delta/2})$ and total messages sent is $W_{2,red} \times (n/b) = mln/b^3 = O(mln^{1-3\delta/2})$. Hence, the total latency and bandwidth costs for blocked multiplication are $O(mln^{1-3\delta/2})$ and $O(mln^{1-\delta/2})$, respectively. This analysis justifies the costs summarized in Table 3.1 and proves the theorem.

3.6.2 Proof of Theorem 4

The following three lemmas will assist us with the proof of Theorem 4.

Lemma 1. *let $\mathbf{S}_c \in \mathbb{R}^{n \times b}$ be a Count sketch matrix. Then, for any vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{n \times 1}$, the following holds*

$$\mathbb{E}[\boldsymbol{x}^T \mathbf{S}_c \mathbf{S}_c^T \boldsymbol{y}] = \boldsymbol{x}^T \boldsymbol{y} \tag{3.10}$$

$$\begin{aligned} Var[\boldsymbol{x}^T \mathbf{S}_c \mathbf{S}_c^T \boldsymbol{y}] &= \frac{1}{b}\left(\sum_{j \neq l} x_j^2 y_l^2 + \sum_{j \neq l} x_j y_j x_l y_l\right) \\ &\leq \frac{1}{b}\left((\boldsymbol{x}^T \boldsymbol{y})^2 + ||\boldsymbol{x}||_2^2 ||\boldsymbol{y}||_2^2\right) \leq \frac{2}{b}||\boldsymbol{x}||_2^2 ||\boldsymbol{y}||_2^2. \end{aligned} \tag{3.11}$$

Proof. See [68], Appendix A. □

Lemma 2. *Let $\mathbf{S} = \frac{1}{\sqrt{N}}(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_N) \in \mathbb{R}^{n \times d}$, where $d = Nb$ and $\mathbf{S}_i \in \mathbb{R}^{n \times b}$ is a Count-sketch matrix that satisfies* (3.10) *and* (3.11)*, for all $i \in 1, 2, \cdots, N$. Then, for any vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{n \times 1}$, the following holds*

$$\begin{aligned} \mathbb{E}[\boldsymbol{x}^T \mathbf{S}\mathbf{S}^T \boldsymbol{y}] &= \boldsymbol{x}^T \boldsymbol{y} \\ Var[\boldsymbol{x}^T \mathbf{S}\mathbf{S}^T \boldsymbol{y}] &\leq \frac{2}{d}||\boldsymbol{x}||_2^2 ||\boldsymbol{y}||_2^2. \end{aligned}$$

Proof. Note that, $\mathbf{S}\mathbf{S}^T = \frac{1}{N}(\mathbf{S}_1\mathbf{S}_1^T + \mathbf{S}_2\mathbf{S}_2^T + \cdots + \mathbf{S}_N\mathbf{S}_N^T)$. Thus,

$$\mathbf{x}^T \mathbf{S}\mathbf{S}^T \mathbf{y} = \frac{1}{N}(\mathbf{x}^T \mathbf{S}_1 \mathbf{S}_1^T \mathbf{y} + \mathbf{x}^T \mathbf{S}_2 \mathbf{S}_2^T \mathbf{y} + \cdots + \mathbf{x}^T \mathbf{S}_N \mathbf{S}_N^T \mathbf{y}),$$

and hence, $\mathbb{E}[\mathbf{x}^T\mathbf{S}\mathbf{S}^T\mathbf{y}] = \mathbf{x}^T\mathbf{y}$ by (3.10) and linearity of expectation. Now,

$$\begin{aligned}
&\text{Var}[\mathbf{x}^T\mathbf{S}\mathbf{S}^T\mathbf{y}] = \mathbb{E}[(\mathbf{x}^T\mathbf{S}\mathbf{S}^T\mathbf{y} - \mathbf{x}^T\mathbf{y})^2] \\
&= \mathbb{E}\Big[\tfrac{1}{N}\left((\mathbf{x}^T\mathbf{S}_1\mathbf{S}_1^T\mathbf{y} + \mathbf{x}^T\mathbf{S}_2\mathbf{S}_2^T\mathbf{y} + \cdots + \mathbf{x}^T\mathbf{S}_N\mathbf{S}_N^T\mathbf{y}) - N\mathbf{x}^T\mathbf{y}\right)^2\Big] \\
&= \mathbb{E}\left[\frac{1}{N^2}\left(\sum_{i=1}^{N}(\mathbf{x}^T\mathbf{S}_i\mathbf{S}_i^T\mathbf{y} - \mathbf{x}^T\mathbf{y})\right)^2\right] \\
&= \frac{1}{N^2}\Bigg(\sum_{i=1}^{N}\mathbb{E}[(\mathbf{x}^T\mathbf{S}_i\mathbf{S}_i^T\mathbf{y} - \mathbf{x}^T\mathbf{y})^2] \\
&\qquad + \sum_{i\neq j}\mathbb{E}[(\mathbf{x}^T\mathbf{S}_i\mathbf{S}_i^T\mathbf{y} - \mathbf{x}^T\mathbf{y})(\mathbf{x}^T\mathbf{S}_j\mathbf{S}_j^T\mathbf{y} - \mathbf{x}^T\mathbf{y})]\Bigg) \\
&= \frac{1}{N^2}\Bigg(\sum_{i=1}^{N}\text{Var}[\mathbf{x}^T\mathbf{S}_i\mathbf{S}_i^T\mathbf{y}] + \\
&\quad \sum_{i\neq j}\mathbb{E}[(\mathbf{x}^T\mathbf{S}_i\mathbf{S}_i^T\mathbf{y} - \mathbf{x}^T\mathbf{y})(\mathbf{x}^T\mathbf{S}_j\mathbf{S}_j^T\mathbf{y} - \mathbf{x}^T\mathbf{y})]\Bigg). \qquad (3.12)
\end{aligned}$$

Noting that $\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_N$ are independent random variables and using (3.10), we get

$$\begin{aligned}
&\mathbb{E}[(\mathbf{x}^T\mathbf{S}_i\mathbf{S}_i^T\mathbf{y} - \mathbf{x}^T\mathbf{y})(\mathbf{x}^T\mathbf{S}_j\mathbf{S}_j^T\mathbf{y} - \mathbf{x}^T\mathbf{y})] \\
&= \mathbb{E}[\mathbf{x}^T\mathbf{S}_i\mathbf{S}_i^T\mathbf{y} - \mathbf{x}^T\mathbf{y}]\mathbb{E}[\mathbf{x}^T\mathbf{S}_j\mathbf{S}_j^T\mathbf{y} - \mathbf{x}^T\mathbf{y}] = 0 \;\forall\; i \neq j.
\end{aligned}$$

Now, using the above equation and (3.11) in (3.12), we get

$$\text{Var}[\mathbf{x}^T\mathbf{S}\mathbf{S}^T\mathbf{y}] = \frac{1}{N^2} \times N \times \frac{2}{b}||\mathbf{x}||_2^2||\mathbf{y}||_2^2 = \frac{2}{d}||\mathbf{x}||_2^2||\mathbf{y}||_2^2,$$

which proves the lemma. □

Lemma 3. *Let $d = 2/\epsilon$. Then, for any $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times l}$ and $\mathbf{S}$ as defined in lemma 2,*

$$\mathbb{E}||\mathbf{A}\mathbf{B} - \mathbf{A}\mathbf{S}\mathbf{S}^T\mathbf{B}||_F^2 \leq \epsilon||\mathbf{A}||_F^2||\mathbf{B}||_F^2. \qquad (3.13)$$

Proof. By the property of Frobenius norm and linearity of expectation, we have

$$\mathbb{E}||\mathbf{A}\mathbf{B} - \mathbf{A}\mathbf{S}\mathbf{S}^T\mathbf{B}||_F^2 = \sum_{i=1}^{m}\sum_{j=1}^{l}\mathbb{E}|\mathbf{a}^{(i)}\mathbf{b}_{(j)} - \mathbf{a}^{(i)}\mathbf{S}\mathbf{S}^T\mathbf{b}_{(j)}|^2, \qquad (3.14)$$

where $\mathbf{a}^{(i)}$ and $\mathbf{b}_{(j)}$ are the i-th row and j-th columns of $\mathbf{A}$ and $\mathbf{B}$, respectively. Now, using lemma 2 in (3.14), we get

$$\begin{aligned}
\mathbb{E}||\mathbf{AB} - \mathbf{ASS}^T\mathbf{B}||_F^2 &= \sum_{i=1}^{m}\sum_{j=1}^{k} \text{Var}[\mathbf{a}^{(i)}\mathbf{SS}^T\mathbf{b}_{(j)}]^2 \\
&\leq \sum_{i=1}^{m}\sum_{j=1}^{k} \frac{2}{d}||\mathbf{a}^{(i)}||_2^2||\mathbf{b}_{(j)}||_2^2 \\
&= \epsilon \left(\sum_{i=1}^{m} ||\mathbf{a}^{(i)}||_2^2\right)\left(\sum_{j=1}^{k} ||\mathbf{b}_{(j)}||_2^2\right) \quad (\text{as } d = 2/\epsilon) \\
&= \epsilon||\mathbf{A}||_F^2||\mathbf{B}||_F^2,
\end{aligned}$$

which is the desired result. □

We are now ready to prove theorem 4. As illustrated in figure 3.4, we can think of computation of a $b \times b$ sub-block $\mathbf{C}(i,j)$ as multiplication of row block $\tilde{\mathbf{A}}(i,:)$ of $\tilde{\mathbf{A}} = \mathbf{AS}$ and column-block $\tilde{\mathbf{B}}(:,j)$ of $\tilde{\mathbf{B}} = \mathbf{S}^T\mathbf{B}$. Since we ignore upto only e workers in the calculation of a $b \times b$ block of $\mathbf{C}$, the effective sketch dimension is greater than $d = \frac{2}{\epsilon\theta}$, and therefore, from lemma 3

$$\begin{aligned}
\mathbb{E}||\mathbf{A}(i,:)\mathbf{B}(:,j) - \mathbf{A}(i,:)\mathbf{S}_{ij}\mathbf{S}_{ij}^T\mathbf{B}(:,j)||_F^2 \\
\leq \epsilon\theta||\mathbf{A}(i,:)||_F^2||\mathbf{B}(:,j)||_F^2, \qquad (3.15)
\end{aligned}$$

for all $i \in 1, \cdots, m/b$ and $j \in 1, \cdots, l/b$. Note that even if we applied the same sketch on $\mathbf{A}$ and $\mathbf{B}$ across row and column blocks, respectively, $\mathbf{S}_{ij}$ in the above equation might end up being different for each pair (i,j) depending upon the location of stragglers, though with a common property that the sketch dimension is at least d. Now, we note that

$$\begin{aligned}
&\mathbb{E}||\mathbf{ASS}^T\mathbf{B} - \mathbf{AB}||_F^2 \\
&= \sum_{i=1}^{m/b}\sum_{j=1}^{l/b} \mathbb{E}||\mathbf{A}(i,:)\mathbf{B}(:,j) - \mathbf{A}(i,:)\mathbf{S}_{ij}\mathbf{S}_{ij}^T\mathbf{B}(:,j)||_F^2 \\
&\leq \epsilon\theta \sum_{i=1}^{m/b}\sum_{j=1}^{l/b} ||\mathbf{A}(i,:)||_F^2||\mathbf{B}(:,j)||_F^2 = \epsilon\theta||\mathbf{A}||_F^2||\mathbf{B}||_F^2.
\end{aligned}$$

Now, by Markov's inequality

$$\begin{aligned}
\mathbb{P}(||\mathbf{ASS}^T\mathbf{B} - \mathbf{AB}||_F^2 > \epsilon||\mathbf{A}||_F^2||\mathbf{B}||_F^2) \\
\leq \frac{\mathbb{E}||\mathbf{ASS}^T\mathbf{B} - \mathbf{AB}||_F^2}{\epsilon||\mathbf{A}||_F^2||\mathbf{B}||_F^2} \\
\leq \frac{\epsilon\theta||\mathbf{A}||_F^2||\mathbf{B}||_F^2}{\epsilon||\mathbf{A}||_F^2||\mathbf{B}||_F^2} = \theta,
\end{aligned}$$

which proves the desired result. □

3.7 Conclusion

Serverless computing penetrates a large user base by allowing users to run distributed applications without the hassles of server management. We analyzed the cost of distributed computation in serverless computing for naive and blocked matrix multiplication. Through analysis and experiments on AWS Lambda, we show that the latter significantly outperforms the former. Thus, existing straggler mitigation schemes that do naive matrix multiplication are unsuitable. To this end, we develop OverSketch, a sketching based algorithm for approximate blocked matrix multiplication. Our sketching scheme requires time linear in the size of input matrices. As a distributed matrix multiplication algorithm, OverSketch has many advantages: reduction in dimension of input matrices for computational savings, and built-in straggler resiliency. Extensive experiments on AWS Lambda support our claims that OverSketch is resilient to stragglers, cost-efficient, and highly accurate for suitably chosen sketch dimension.

Part II

Speeding up Convex Optimization

In this part of the thesis, we study algorithms for speeding up convex optimization in the serverless setting.

- **Chapter 4:** Using sketching methods with large-number of slow workers
- **Chapter 5:** Using local optimization methods with large-number of slow workers
- **Chapter 6:** Using sketching methods for limited memory optimization

Chapter 4

OverSketched Newton

In this chapter, we develop OverSketched Newton, a randomized Hessian-based optimization algorithm to solve large-scale convex optimization problems in serverless systems.

4.1 Introduction

As outlined in Chapter 3, there are several crucial differences between the traditional High Performance Computing (HPC) / serverful and serverless architectures, such as scale, memory, mode of communication, system noise, etc. Due to these reasons, existing distributed algorithms cannot, in general, be extended to serverless computing. For example, first-order methods for optimization, such as gradient descent and Nesterov Accelerated Gradient (NAG) methods, tend to perform poorly on distributed serverless architectures [4]. Their slower convergence is made worse on serverless platforms due to persistent stragglers. The straggler effect incurs heavy slowdown due to the accumulation of tail times as a result of a subset of slow workers occurring in each iteration.

Compared to first-order optimization algorithms, second-order methods—which use the gradient as well as Hessian information—enjoy superior convergence rates. For instance, Newton's method enjoys quadratic convergence for strongly convex and smooth problems, compared to the linear convergence of gradient descent [140]. Moreover, second-order methods do not require step-size tuning and unit step-size provably works for most problems. These methods have a long history in optimization and scientific computing (see, e.g., [140]), but they are less common in machine learning and data science. This is partly since stochastic first order methods suffice for downstream problems [141] and partly since naive implementations of second order methods can perform poorly [142]. However, recent theoretical work has addressed many of these issues [42], [44]–[46], [143], and recent implementations have shown that high-quality implementations of second order stochastic optimization algorithms can beat state-of-the-art in machine learning applications [51], [66], [67], [144], [145] in traditional systems.

4.1.1 Contributions

In this chapter, we argue that second-order methods are highly compatible with serverless systems that provide extensive computing power by invoking thousands of workers but are limited by the communication costs and hence the number of iterations; and, to address the challenges of ephemeral workers and stragglers in serverless systems, we propose and analyze a randomized and distributed second-order optimization algorithm, called *OverSketched Newton*. OverSketched Newton uses the technique of matrix sketching from Sub-Sampled Newton (SSN) methods [44]–[46], [143], which are based on sketching methods from Randomized Numerical Linear Algebra (RandNLA) [41], [47], [146], to obtain a good approximation for the Hessian, instead of calculating the full Hessian.

OverSketched Newton has two key components. For straggler-resilient Hessian calculation in serverless systems, we use the sparse sketching based randomized matrix multiplication method from [5]. For straggler mitigation during gradient calculation, we use the recently proposed technique based on error-correcting codes to create redundant computation [24], [27], [147]. We prove that, for strongly convex functions, the local convergence rate of OverSketched Newton is linear-quadratic, while its global convergence rate is linear. Then, going beyond the usual strong convexity assumption for second-order methods, we adapt OverSketched Newton using ideas from [143]. For such functions, we prove that a linear convergence rate can be guaranteed with OverSketched Newton. To the best of our knowledge, this is the first work to prove convergence guarantees for weakly-convex problems when the Hessian is computed approximately using ideas from RandNLA.

We extensively evaluate OverSketched Newton on AWS Lambda using several real-world datasets obtained from the LIBSVM repository [119], and we compare OverSketched Newton with several first-order (gradient descent, Nesterov's method, etc.) and second-order (exact Newton's method [140], GIANT [51], etc.) baselines for distributed optimization. We further evaluate and compare different techniques for straggler mitigation, such as speculative execution, coded computing [24], [27], randomization-based sketching [5] and gradient coding [25]. We demonstrate that OverSketched Newton is at least 9x and 2x faster than state-of-the-art first-order and second-order schemes, respectively, in terms of end-to-end training time on AWS Lambda. Moreover, we show that OverSketched Newton on serverless systems outperforms existing distributed optimization algorithms in serverful systems by at least 30%.

To the best of our knowledge, this is the first work that proposes a large-scale distributed optimization algorithm that specifically caters to *serverless architectures* with *provable convergence guarantees*. We exploit the advantages offered by serverless systems while mitigating the drawbacks such as stragglers and additional overhead per invocation of workers.

4.2 Newton's Method: An Overview

We are interested in solving on serverless systems in a distributed and straggler-resilient manner problems of the form:

$$f(\mathbf{w}^*) = \min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}), \tag{4.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a closed and convex function bounded from below. In the Newton's method, the update at the $(t+1)$-th iteration is obtained by minimizing the Taylor's expansion of the objective function $f(\cdot)$ at $\mathbf{w}_t$, that is

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \Big\{ f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^T(\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^T \nabla^2 f(\mathbf{w}_t)(\mathbf{w} - \mathbf{w}_t) \Big\}. \tag{4.2}$$

For strongly convex $f(\cdot)$, that is, when $\nabla^2 f(\cdot)$ is invertible, Eq. (4.2) becomes $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1}\nabla f(\mathbf{w}_t)$, where $\mathbf{H}_t = \nabla^2 f(\mathbf{w}_t)$ is the Hessian matrix at the t-th iteration. Given a good initialization and assuming that the Hessian is Lipschitz, the Newton's method satisfies the update $||\mathbf{w}_{t+1} - \mathbf{w}^*||_2 \leq c||\mathbf{w}_t - \mathbf{w}^*||_2^2$, for some constant $c > 0$, implying quadratic convergence [140].

One shortcoming for the classical Newton's method is that it works only for strongly convex objective functions. In particular, if f is weakly-convex[1], that is, if the Hessian matrix is not positive definite, then the objective function in (4.2) may be unbounded from below. To address this shortcoming, authors in [143] recently proposed a variant of Newton's method, called Newton-Minimum-Residual (Newton-MR). Instead of (4.1), Newton-MR considers the following auxiliary optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} ||\nabla f(\mathbf{w})||^2.$$

Note that the minimizers of this auxiliary problem and (4.1) are the same when $f(\cdot)$ is convex. Then, the update direction in the $(t+1)$-th iteration is obtained by minimizing the Taylor's expansion of $||\nabla f(\mathbf{w}_t + \mathbf{p})||^2$, that is,

$$\mathbf{p}_t = \arg\min_{\mathbf{w}\in\mathbb{R}^d} ||\nabla f(\mathbf{w}_t) + \mathbf{H}_t\mathbf{p}||^2.$$

The general solution of the above problem is given by $\mathbf{p} = -[\mathbf{H}_t]^\dagger \nabla f(\mathbf{w}_t) + (\mathbf{I} - \mathbf{H}_t[\mathbf{H}_t]^\dagger)\mathbf{q},\ \forall\, \mathbf{q} \in \mathbb{R}^d$, where $[\cdot]^\dagger$ is the Moore-Penrose inverse. Among these, the minimum norm solution is chosen, which gives the update direction in the t-th iteration as $\mathbf{p}_t = -\mathbf{H}_t^\dagger \nabla f(\mathbf{w}_t)$. Thus, the model update is

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{p}_t = \mathbf{w}_t - [\nabla^2 f(\mathbf{w}_t)]^\dagger \nabla f(\mathbf{w}_t). \tag{4.3}$$

OverSketched Newton considers both of these variants.

4.3 OverSketched Newton

In many applications like machine learning where the training data itself is noisy, using the exact Hessian is not necessary. Indeed, many results in the literature prove convergence guarantees for Newton's method when the Hessian is computed approximately using ideas from RandNLA for a single machine (e.g. [42], [44], [45], [148]). In particular, these methods perform a form of dimensionality reduction for the Hessian using random matrices, called sketching matrices. Many popular sketching schemes have been proposed in the literature, for example, sub-Gaussian, Hadamard, random row sampling, sparse Johnson-Lindenstrauss, etc. [41], [47]. Inspired from these works, we present OverSketched Newton, a stochastic second order algorithm for solving—*on serverless systems, in a distributed, straggler-resilient manner*—problems of the form (4.1).

[1]For the sake of clarity, we call a convex function weakly-convex if it is not strongly convex.

Algorithm 8: Straggler-resilient distributed computation of $\mathbf{Ax}$ using codes

Input : Matrix $\mathbf{A} \in \mathbb{R}^{t \times s}$, vector $x \in \mathbb{R}^s$, and block size parameter b

Result: $\mathbf{y} = \mathbf{Ax}$, where $\mathbf{y} \in \mathbb{R}^s$ is the product of matrix $\mathbf{A}$ and vector $\mathbf{x}$

Initialization: Divide $\mathbf{A}$ into $T = t/b$ row-blocks, each of dimension $b \times s$

Encoding: Generate coded $\mathbf{A}$, say $\mathbf{A}_c$, in parallel using a 2D product code by arranging the row blocks of $\mathbf{A}$ in a 2D structure of dimension $\sqrt{T} \times \sqrt{T}$ and adding blocks across rows and columns to generate parities; see Fig. 2 in [27] for an illustration

for $i = 1$ *to* $T + 2\sqrt{T} + 1$ **do**

1. Worker W_i receives the i-th row-block of $\mathbf{A}_c$, say $\mathbf{A}_c(i, :)$, and $\mathbf{x}$ from cloud storage
2. W_i computes $\mathbf{y}(i) = \mathbf{A}(i, :)\mathbf{x}$
3. Master receives $\mathbf{y}(i)$ from worker W_i

end

Decoding: Master checks if it has received results from enough workers to reconstruct $\mathbf{y}$. Once it does, it decodes $\mathbf{y}$ from available results using the peeling decoder

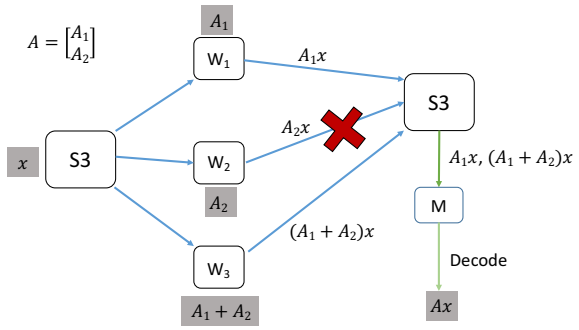


Figure 4.1: **Coded matrix-vector multiplication**: Matrix $\mathbf{A}$ is divided into 2 row chunks $\mathbf{A}_1$ and $\mathbf{A}_2$. During encoding, redundant chunk $\mathbf{A}_1 + \mathbf{A}_2$ is created. Three workers obtain $\mathbf{A}_1$,$\mathbf{A}_2$ and $\mathbf{A}_1 + \mathbf{A}_2$ from the cloud storage S3, respectively, and then multiply by $\mathbf{x}$ and write back the result to the cloud. The master M can decode $\mathbf{Ax}$ from the results of any two workers, thus being resilient to one straggler (W_2 in this case).

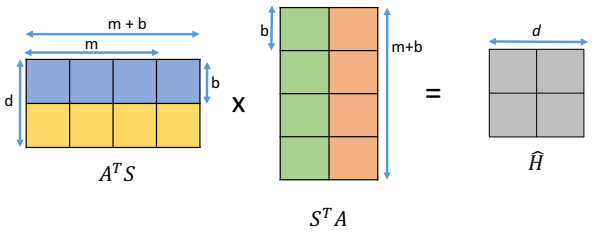


Figure 4.2: **OverSketch-based approximate Hessian computation:** First, the matrix $\mathbf{A}$—satisfying $\mathbf{A}^T\mathbf{A} = \nabla^2 f(\mathbf{w}_t)$—is sketched in parallel using the sketch in (4.4). Then, each worker receives a block of each of the sketched matrices $\mathbf{A}^T\mathbf{S}$ and $\mathbf{S}^T\mathbf{A}$, multiplies them, and communicates back its results for reduction. During reduction, stragglers can be ignored by the virtue of "over" sketching. For example, here the desired sketch dimension m is increased by block-size b for obtaining resiliency against one straggler for each block of $\hat{\mathbf{H}}$.

Distributed straggler-resilient gradient computation: OverSketched Newton computes the full gradient in each iteration by using tools from error-correcting codes [24], [27]. Our key observation is that, for several commonly encountered optimization problems, gradient computation relies on matrix-vector multiplications (see Sec. 4.4 for examples). We leverage coded matrix multiplication technique from [27] to perform the large-scale matrix-vector multiplication in a distributed straggler-resilient manner. The idea of coded matrix multiplication is explained in Fig. 4.1; detailed steps are provided in Algorithm 8.

Distributed straggler-resilient approximate Hessian computation: For several commonly encountered optimization problems, Hessian computation involves matrix-matrix multiplication for a pair of large matrices (see Sec. 4.4 for several examples). For computing the large-scale matrix-matrix multiplication in parallel in serverless systems, we propose to use a straggler-resilient scheme called *OverSketch* from [5]. OverSketch does *blocked partitioning* of input matrices where each worker works on square blocks of dimension b. Hence, it is more communication efficient than existing coding-based straggler mitigation schemes that do naïve row-column partition of input matrices [26], [28]. We note that it is well known in HPC that blocked partitioning of input matrices can lead to communication-efficient methods for distributed multiplication [5], [113], [114].

OverSketch uses a sparse sketching matrix based on Count-Sketch [41]. It has similar computational efficiency and accuracy guarantees as that of the Count-Sketch, with two additional properties: it is amenable to distributed implementation; and it is resilient to stragglers. We briefly review the construction of the OverSketch matrix from Chapter 3 for readers' convenience.

Recall that the Hessian $\nabla^2 f(\cdot) \in \mathbb{R}^{d\times d}$. First choose the desired sketch dimension m (which depends on d), block-size b (which depends on the memory of the workers), and straggler tolerance $\zeta > 0$ (which depends on the distributed system). Then, define $N = m/b$ and $e = \zeta N$, for some constant $\zeta > 0$. Here ζ is the fraction of stragglers that we want our algorithm to tolerate. Thus, e is the maximum number of stragglers per $N + e$ workers that can be tolerated. The sketch $\mathbf{S}$ is then given by

$$\mathbf{S} = \frac{1}{\sqrt{N}}(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N+e}), \tag{4.4}$$

where $\mathbf{S}_i \in \mathbb{R}^{n\times b}$, for all $i \in [1, N + e]$, are i.i.d. Count-Sketch matrices[2] with sketch dimension b. Note that $\mathbf{S} \in \mathbb{R}^{n\times(m+eb)}$, where $m = Nb$ is the required sketch dimension and e is the over-provisioning parameter to provide resiliency against e stragglers per $N+e$ workers. We leverage the straggler resiliency of OverSketch to obtain the sketched Hessian in a distributed straggler-resilient manner. An illustration of OverSketch is provided in Fig. 4.2; see Algorithm 9 for details.

Model update: Let $\hat{\mathbf{H}}_t = \mathbf{A}_t^T \mathbf{S}_t \mathbf{S}_t^T \mathbf{A}_t$, where $\mathbf{A}_t$ is the square root of the Hessian $\nabla^2 f(\mathbf{w}_t)$, and $\mathbf{S}_t$ is an independent realization of (4.4) at the t-th iteration. For strongly-convex functions, the update direction is $\mathbf{p}_t = -\hat{\mathbf{H}}_t^{-1}\nabla f(\mathbf{w}_t)$. We use line-search to choose the step-size, that is, find

$$\alpha_t = \max_{\alpha\leq 1} \alpha \quad \text{such that} \quad f(\mathbf{w}_t + \alpha\mathbf{p}_t) \leq f(\mathbf{w}_t) + \alpha\beta\mathbf{p}_t^T\nabla f(\mathbf{w}_t), \tag{4.5}$$

for some constant $\beta \in (0, 1/2]$. For weakly-convex functions, the update direction (inspired by Newton-MR [143]) is $\mathbf{p}_t = -\hat{\mathbf{H}}_t^{\dagger}\nabla f(\mathbf{w}_t)$, where $\hat{\mathbf{H}}_t^{\dagger}$ is the Moore-Penrose inverse of $\hat{\mathbf{H}}_t$. To find the update $\mathbf{w}_{t+1}$, we find the right step-size α_t using line-search in (4.5), but with $f(\cdot)$ replaced by $||\nabla f(\cdot)||^2$ and $\nabla f(\mathbf{w}_t)$ replaced by $2\hat{\mathbf{H}}_t\nabla f(\mathbf{w}_t)$, according to the objective in $||\nabla f(\cdot)||^2$. More specifically, for some constant $\beta \in (0, 1/2]$,

$$\alpha_t = \max_{\alpha\leq 1} \alpha \quad \text{such that} \quad ||\nabla f(\mathbf{w}_t + \alpha\mathbf{p}_t)||^2 \leq ||\nabla f(\mathbf{w}_t)||^2 + 2\alpha\beta\mathbf{p}_t^T\hat{\mathbf{H}}_t\nabla f(\mathbf{w}_t). \tag{4.6}$$

[2]Each of the Count-Sketch matrices $\mathbf{S}_i$ is constructed (independently of others) as follows. First, for every row j, $j \in [n]$, of $\mathbf{S}_i$, independently choose a column $h(j) \in [b]$. Then, select a uniformly random element from $\{-1, +1\}$, denoted as $\sigma(i)$. Finally, set $\mathbf{S}_i(j, h(j)) = \sigma(i)$ and set $\mathbf{S}_i(j, l) = 0$ for all $l \neq h(j)$. (See [5], [41] for details.)

Algorithm 9: Approximate Hessian calculation on serverless systems using OverSketch

Input : Matrices $\mathbf{A} \in \mathbb{R}^{n \times d}$, required sketch dimension m, straggler tolerance e, block-size b. Define $N = m/b$

Result: $\hat{\mathbf{H}} \approx \mathbf{A}^T \times \mathbf{A}$

1 **Sketching**: Use sketch in Eq. (4.4) to obtain $\tilde{\mathbf{A}} = \mathbf{S}^T\mathbf{A}$ distributedly (see Algorithm 5 in [5] for details)

2 **Block partitioning**: Divide $\tilde{\mathbf{A}}$ into $(N + e) \times d/b$ matrix of $b \times b$ blocks

3 **Computation phase**: Each worker takes a block of $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{A}}^T$ each and multiplies them. This step invokes $(N + e)d^2/b^2$ workers, where $N + e$ workers compute one block of $\hat{\mathbf{H}}$

4 **Termination**: Stop computation when any N out of $N + e$ workers return their results for each block of $\hat{\mathbf{H}}$

5 **Reduction phase**: Invoke d^2/b^2 workers to aggregate results during the computation phase, where each worker will calculate one block of $\hat{\mathbf{H}}$

Algorithm 10: OverSketched Newton in a nutshell

Input : Convex function f; Initial iterate $\mathbf{w}_0 \in \mathbb{R}^d$; Line search parameter $0 < \beta \leq 1/2$; Number of iterations T

1 **for** $t = 1$ *to* T **do**

2 Compute full gradient $\mathbf{g}_t$ in a distributed fashion using Algorithm 8

3 Compute sketched Hessian matrix $\hat{\mathbf{H}}_t$ in a distributed fashion using Algorithm 9

4 **if** f *is strongly-convex* **then**

5 Compute the update direction at the master as: $\mathbf{p}_t = -[\hat{\mathbf{H}}_t]^{-1}\nabla f(\mathbf{w}_t)$

6 Compute step-size α_t satisfying the line-search condition (4.5) in a distributed fashion

7 **else**

8 Compute the update direction at the master as: $\mathbf{p}_t = -[\hat{\mathbf{H}}_t]^{\dagger}\nabla f(\mathbf{w}_t)$

9 Find step-size α_t satisfying the line-search condition (4.6) in a distributed fashion

10 **end**

11 Compute the model update $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t\mathbf{p}_t$ at the master

12 **end**

Note that for OverSketched Newton, we use $\hat{\mathbf{H}}_t$ in the line-search since the exact Hessian is not available. The update in the t-th iteration in both cases is given by

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t\mathbf{p}_t.$$

Note that (4.5) line-search can be solved approximately in single machine systems using Armijo backtracking line search [140], [149]. OverSketched Newton is concisely described in Algorithm 10. In Section 4.3.2, we describe how to implement distributed line-search in serverless systems

when the data is stored in the cloud. Next, we prove convergence guarantees for OverSketched Newton that uses the sketch matrix in (4.4) and full gradient for approximate Hessian computation.

4.3.1 Convergence Guarantees

First, we focus our attention to strongly convex functions. We consider the following assumptions. We note that these assumptions are standard for analyzing approximate Newton methods, (e.g., see [42], [44], [45].

Assumptions:

1. f is twice-differentiable;
2. f is k-strongly convex ($k > 0$), that is,
$$\nabla^2 f(\mathbf{w}) \succeq k\mathbf{I};$$
3. f is M-smooth ($k \leq M < \infty$), that is,
$$\nabla^2 f(\mathbf{w}) \preceq M\mathbf{I};$$
4. the Hessian is L-Lipschitz continuous, that is, for any $\mathbf{\Delta} \in \mathbb{R}^d$
$$||\nabla^2 f(\mathbf{w} + \mathbf{\Delta}) - \nabla^2 f(\mathbf{w})||_2 \leq L||\mathbf{\Delta}||_2,$$
where $||\cdot||_2$ is the spectral norm for matrices.

We first prove the following "global" convergence guarantee which shows that OverSketched Newton would converge from any random initialization of $\mathbf{w}_0 \in \mathbb{R}^d$ with high probability.

Theorem 5 (Global convergence for strongly-convex f). *Consider Assumptions 1, 2, and 3 and step-size α_t given by Eq. (4.5). Let $\mathbf{w}^*$ be the optimal solution of (4.1). Let ϵ and μ be positive constants. Then, using the sketch in (4.4) with a sketch dimension $Nb + eb = \Omega(\frac{d^{1+\mu}}{\epsilon^2})$ and the number of column-blocks $N + e = \Theta_\mu(1/\epsilon)$, the updates for OverSketched Newton, for any $\mathbf{w}_t \in \mathbb{R}^d$, satisfy*
$$f(\mathbf{w}_{t+1}) - f(\mathbf{w}^*) \leq (1 - \rho)(f(\mathbf{w}_t) - f(\mathbf{w}^*)),$$
with probability at least $1 - 1/d^\tau$, where $\rho = \frac{2\alpha_t\beta k}{M(1+\epsilon)}$ and $\tau > 0$ is a constant depending on μ and constants in $\Omega(\cdot)$ and $\Theta(\cdot)$. Moreover, α_t satisfies $\alpha_t \geq \frac{2(1-\beta)(1-\epsilon)k}{M}$.

Proof. See Section 4.6.1. □

Theorem 5 guarantees the global convergence of OverSketched Newton starting with any initial estimate $\mathbf{w}_0 \in \mathbb{R}^d$ to the optimal solution $\mathbf{w}^*$ with at least a linear rate.

Next, we can also prove an additional "local" convergence guarantee for OverSketched Newton, under the assumption that $\mathbf{w}_0$ is sufficiently close to $\mathbf{w}^*$.

Theorem 6 (**Local convergence for strongly-convex** f)**.** *Consider Assumptions 1, 2, and 4 and step-size* $\alpha_t = 1$*. Let* $\mathbf{w}^*$ *be the optimal solution of* (4.1) *and* γ *and* β *be the minimum and maximum eigenvalues of* $\nabla^2 f(\mathbf{w}^*)$*, respectively. Let* $\epsilon \in (0, \gamma/(8\beta)]$ *and* $\mu > 0$*. Then, using the sketch in* (4.4) *with a sketch dimension* $Nb + eb = \Omega(\frac{d^{1+\mu}}{\epsilon^2})$ *and the number of column-blocks* $N + e = \Theta_\mu(1/\epsilon)$*, the updates for OverSketched Newton, with initialization* $\mathbf{w}_0$ *such that* $||\mathbf{w}_0 - \mathbf{w}^*||_2 \leq \frac{\gamma}{8L}$*, follow*

$$||\mathbf{w}_{t+1} - \mathbf{w}^*||_2 \leq \frac{25L}{8\gamma}||\mathbf{w}_t - \mathbf{w}^*||_2^2 + \frac{5\epsilon\beta}{\gamma}||\mathbf{w}_t - \mathbf{w}^*||_2 \quad \textit{for } t = 1, 2, \cdots, T,$$

with probability at least $1 - T/d^\tau$*, where* $\tau > 0$ *is a constant depending on* μ *and constants in* $\Omega(\cdot)$ *and* $\Theta(\cdot)$*.*

Proof. See Section 4.6.2. □

Theorem 6 implies that the convergence is linear-quadratic in error $\Delta_t = \mathbf{w}_t - \mathbf{w}^*$. Initially, when $||\Delta_t||_2$ is large, the first term of the RHS will dominate and the convergence will be quadratic, that is, $||\Delta_{t+1}||_2 \lesssim \frac{25L}{8\gamma}||\Delta_t||_2^2$. In later stages, when $||\mathbf{w}_t - \mathbf{w}^*||_2$ becomes sufficiently small, the second term of RHS will start to dominate and the convergence will be linear, that is, $||\Delta_{t+1}||_2 \lesssim \frac{5\epsilon\beta}{\gamma}||\Delta_t||_2$. At this stage, the sketch dimension can be increased to reduce ϵ to diminish the effect of the linear term and improve the convergence rate in practice. Note that, for second order methods, the number of iterations T is in the order of tens in general while the number of features d is typically in thousands. Hence, the probability of failure is generally small (and can be made negligible by choosing τ appropriately).

Though the works [42], [44], [45], [148], [150] also prove convergence guarantees for approximate Hessian-based optimization, no convergence results exist for the OverSketch matrix in Eq. (4.4) to the best of our knowledge. OverSketch has many nice properties like sparsity, input obliviousness, and amenability to distributed implementation, and our convergence guarantees take into account the block-size b (that captures the amount of communication at each worker) and the number of stragglers e, both of which are a property of the distributed system. On the other hand, algorithms in [42], [44], [45], [148], [150] are tailored to run on a single machine.

Next, we consider the case of weakly-convex functions. For this case, we consider two more assumptions on the Hessian matrix, similar to [143]. These assumptions are a relaxation of the strongly-convex case.

Assumptions:

5. There exists some $\eta > 0$ such that, $\forall\ \mathbf{w} \in \mathbb{R}^d$,

$$||(\nabla^2 f(\mathbf{w}))^\dagger||_2 \leq 1/\eta.$$

This assumption establishes regularity on the pseudo-inverse of $\nabla^2 f(\mathbf{x})$. It also implies that $||\nabla^2 f(\mathbf{w})\mathbf{p}|| \geq \eta||\mathbf{p}||\ \forall\, p \in$ Range$(\nabla^2 f(\mathbf{w}))$, that is, the minimum 'non-zero' eigenvalue of $\nabla^2 f(\mathbf{w})$ is lower bounded by η; just as in the k-strongly convex case, the smallest eigenvalue is greater than k.

6. Let $\mathbf{U} \in \mathbb{R}^{d \times d}$ be any arbitrary orthogonal basis for Range$(\nabla^2 f(\mathbf{w}))$, there exists $0 < \nu \leq 1$, such that,
$$||\mathbf{U^T} \nabla \mathbf{f}(\mathbf{w})||^{\mathbf{2}} \geq \nu ||\nabla \mathbf{f}(\mathbf{w})||^{\mathbf{2}} \ \forall \ \mathbf{w} \in \mathbb{R}^{\mathbf{d}}.$$
This assumption ensures that there is always a non-zero component of the gradient in the subspace spanned by the Hessian, and, thus, ensures that the model update $-\hat{\mathbf{H}}_t^{\dagger} \nabla f(\mathbf{w}_t)$ will not be zero.

Note that the above assumptions are always satisfied by strongly-convex functions. Next, we prove global convergence of OverSketched Newton when the objective is weakly-convex.

Theorem 7 (Global convergence for weakly-convex f). *Consider Assumptions 1,3,4,5 and 6 and step-size α_t given by Eq. (4.6). Let $\epsilon \in \left(0, \frac{(1-\beta)\nu\eta}{2M}\right]$ and $\mu > 0$. Then, using an OverSketch matrix with a sketch dimension $Nb + eb = \Omega(\frac{d^{1+\mu}}{\epsilon^2})$ and the number of column-blocks $N + e = \Theta_\mu(1/\epsilon)$, the updates for OverSketched Newton, for any $\mathbf{w}_t \in \mathbb{R}^d$, satisfy*
$$||\nabla f(\mathbf{w}_{t+1})||^2 \leq \left(1 - 2\beta\alpha\nu \frac{(1-\epsilon)\eta}{M(1+\epsilon)}\right) ||\nabla f(\mathbf{w}_t)||^2,$$
with probability at least $1 - 1/d^\tau$, where $\alpha = \frac{\eta}{2Q}\big[(1-\beta)\nu\eta - 2\epsilon M\big], Q = (L||\nabla f(\mathbf{w}_0)|| + M^2)$, $\mathbf{w}_0$ is the initial iterate of the algorithm and $\tau > 0$ is a constant depending on μ and constants in $\Omega(\cdot)$ and $\Theta(\cdot)$.

Proof. See Section 4.6.3. □

Even though we present the above guarantees for the sketch matrix in Eq. (4.4), our analysis is valid for any sketch that satisfies the *subspace embedding* property (Lemma 4; see [41] for details on subspace embedding property of sketches). To the best of our knowledge, this is the first work to prove the convergence guarantees for weakly-convex functions when the Hessian is calculated approximately using sketching techniques. Later, authors in [151] extended the analysis to the case of general Hessian perturbations with additional assumptions on the type of perturbation.

4.3.2 Distributed Line Search

Here, we describe a line-search procedure for distributed serverless optimization, which is inspired by the line-search method from [51] for serverful systems. To solve for the step-size α_t as described in the optimization problem in (4.5), we set $\beta = 0.1$ and choose a candidate set $\mathcal{S} = \{4^0, 4^1, \cdots, 4^{-5}\}$. After the master calculates the descent direction $\mathbf{p}_t$ in the t-th iteration, the i-th worker calculates $f_i(\mathbf{w}_t + \alpha \mathbf{p}_t)$ for all values of α in the candidate set $\mathcal{S}$, where $f_i(\cdot)$ depends on the local data available at the i-th worker and $f(\cdot) = \sum_i f_i(\cdot)$[3].

[3]For the weakly-convex case, the workers calculate $\nabla f_i(\cdot)$ instead of $f_i(\cdot)$, and the master calculates $||\nabla f(\cdot)||^2$ instead of $f(\cdot)$.

The master then sums the results from workers to obtain $f(\mathbf{w}_t + \alpha \mathbf{p}_t)$ for all values of α in $\mathcal{S}$ and finds the largest α that satisfies the Armijo condition in (4.5)[4]. Note that line search requires an additional round of communication where the master communicates $\mathbf{p}_t$ to the workers through cloud and the workers send back the function values $f_i(\cdot)$. Finally, the master finds the best step-size from set $\mathcal{S}$ and finds the model estimate $\mathbf{w}_{t+1}$.

4.4 OverSketched Newton on Serverless Systems: Examples

Here, we describe several examples where our general approach can be applied.

4.4.1 Logistic Regression using OverSketched Newton

The optimization problem for supervised learning using Logistic Regression takes the form

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \right\}. \tag{4.7}$$

Here, $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^{d\times 1}$ and $y_1, \cdots, y_n \in \mathbb{R}$ are training sample vectors and labels, respectively. The goal is to learn the feature vector $\mathbf{w}^* \in \mathbb{R}^{d\times 1}$. Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ and $\mathbf{y} = [y_1, \cdots, y_n] \in \mathbb{R}^{n\times 1}$ be the example and label matrices, respectively. The gradient for the problem in (4.7) is given by

$$\nabla f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i \mathbf{x}_i}{1 + e^{y_i \mathbf{w}_i^T \mathbf{x}_i}} + \lambda \mathbf{w}.$$

Calculation of $\nabla f(\mathbf{w})$ involves two matrix-vector products, $\boldsymbol{\alpha} = \mathbf{X}^T\mathbf{w}$ and $\nabla f(\mathbf{w}) = \frac{1}{n}\mathbf{X}\boldsymbol{\beta} + \lambda\mathbf{w}$, where $\beta_i = \frac{-y_i}{1+e^{y_i\alpha_i}}\ \forall\ i \in [1, \cdots, n]$. When the example matrix is large, these matrix-vector products are performed distributedly using codes. Faster convergence is obtained by second-order methods which will additionally compute the Hessian $\mathbf{H} = \frac{1}{n}\mathbf{X}\mathbf{\Lambda}\mathbf{X}^T + \lambda\mathbf{I}_d$, where $\mathbf{\Lambda}$ is a diagonal matrix with entries given by $\Lambda(i,i) = \frac{e^{y_i\alpha_i}}{(1+e^{y_i\alpha_i})^2}$. The product $\mathbf{X}\mathbf{\Lambda}\mathbf{X}^T$ is computed approximately in a distributed straggler-resilient manner using the sketch matrix in (4.4). Using the result of distributed multiplication, the Hessian matrix $\mathbf{H}$ is calculated at the master and the model is updated as $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}^{-1}\nabla f(\mathbf{w}_t)$. In practice, efficient algorithm like conjugate gradient, that provide a good estimate in a small number of iterations, can be used locally at the master to solve for $\mathbf{w}_{t+1}$ [152].[5]

We provide a detailed description of OverSketched Newton for large-scale logistic regression for serverless systems in Algorithm 11. Steps 4, 8, and 14 of the algorithm are computed in parallel

[4]Note that codes can be used to mitigate stragglers during distributed line-search in a manner similar to the gradient computation phase.

[5]Note that here we have assumed that the number of features is small enough to perform the model update locally at the master. This is not necessary, and straggler resilient schemes, such as in [147], can be used to perform distributed conjugate gradient in serverless systems.

Algorithm 11: OverSketched Newton: Logistic Regression for Serverless Computing

1 **Input Data** (stored in cloud storage): Example Matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$ and vector $\mathbf{y} \in \mathbb{R}^{n \times 1}$ (stored in cloud storage), regularization parameter λ, number of iterations T, Sketch $\mathbf{S}$ as defined in Eq. (4.4)

2 **Initialization**: Define $\mathbf{w}^1 = \mathbf{0}^{d \times 1}, \boldsymbol{\beta} = \mathbf{0}^{n \times 1}, \boldsymbol{\gamma} = \mathbf{0}^{n \times 1}$, Encode $\mathbf{X}$ and $\mathbf{X}^T$ as described in Algorithm 8

3 **for** $t = 1$ *to* T **do**

4 $\quad \boldsymbol{\alpha} = \mathbf{X}\mathbf{w}^t$; // Compute in parallel using Algorithm 8

5 $\quad$ **for** $i = 1$ *to* n **do**

6 $\quad\quad \beta_i = \frac{-y_i}{1+e^{y_i \alpha_i}}$;

7 $\quad$ **end**

8 $\quad \mathbf{g} = \mathbf{X}^T \boldsymbol{\beta}$; // Compute in parallel using Algorithm 8

9 $\quad \nabla f(\mathbf{w}^t) = \mathbf{g} + \lambda \mathbf{w}^t$;

10 $\quad$ **for** $i = 1$ *to* n **do**

11 $\quad\quad \gamma(i) = \frac{e^{y_i \alpha_i}}{(1+e^{y_i \alpha_i})^2}$;

12 $\quad$ **end**

13 $\quad \mathbf{A} = \sqrt{diag(\boldsymbol{\gamma})}\mathbf{X}^T$

14 $\quad \hat{\mathbf{H}} = \mathbf{A}^T \mathbf{S}\mathbf{S}^T \mathbf{A}$; // Compute in parallel using Algorithm 9

15 $\quad \mathbf{H} = \frac{1}{n}\hat{\mathbf{H}} + \lambda \mathbf{I}_d$;

16 $\quad \mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{H}^{-1} \nabla f(\mathbf{w}^t)$;

17 **end**

Result: $\mathbf{w}^* = \mathbf{w}_{T+1}$

on AWS Lambda. All other steps are simple vector operations that can be performed locally at the master, for instance, the user's laptop. Steps 4 and 8 are executed in a straggler-resilient fashion using the coding scheme in [27], as illustrated in Fig. 1.2 and described in detail in Algorithm 8.

We use the coding scheme in [27] since the encoding can be implemented in parallel and requires less communication per worker compared to the other schemes, for example schemes in [24], [28], that use Maximum Distance Separable (MDS) codes. Moreover, the decoding scheme takes linear time and is applicable on real-valued matrices. Note that since the example matrix $\mathbf{X}$ is constant in this example, the encoding of $\mathbf{X}$ is done only once before starting the optimization algorithm. Thus, the encoding cost can be amortized over iterations. Moreover, decoding over the resultant product vector requires negligible time and space, even when n is scaling into the millions.

The same is, however, not true for the matrix multiplication for Hessian calculation (step 14 of Algorithm 11), as the matrix $\mathbf{A}$ changes in each iteration, thus encoding costs will be incurred in every iteration if error-correcting codes are used. Moreover, encoding and decoding a huge matrix stored in the cloud incurs heavy communication cost and becomes prohibitive. Motivated by this, we use OverSketch in step 14, as described in Algorithm 9, to calculate an approximate matrix multiplication, and hence the Hessian, efficiently in serverless systems with inbuilt straggler

resiliency.[6]

4.4.2 Softmax Regression using OverSketched Newton

We take unregulairzed softmax regression as an illustrative example for the weakly convex case. The goal is to find the weight matrix $\mathbf{W} = [\mathbf{w}_1, \cdots, \mathbf{w}_K]$ that fit the training data $\mathbf{X} \in \mathbb{R}^{d \times N}$ and $\mathbf{y} \in \mathbb{R}^{K \times N}$. Here $\mathbf{w}_i \in \mathbb{R}^d$ represesents the weight vector for the k-th class for all $i \in [1, K]$ and K is the total number of classes. Hence, the resultant feature dimension for softmax regression is dK. The optimization problem is of the form

$$f(\mathbf{W}) = \sum_{n=1}^{N} \left[\sum_{k=1}^{K} y_{kn} \mathbf{w}_k^T \mathbf{x}_n - \log \sum_{l=1}^{K} \exp\left(\mathbf{w}_l^T \mathbf{x}_n\right) \right]. \tag{4.8}$$

The gradient vector for the i-th class is given by

$$\nabla f_i(\mathbf{W}) = \sum_{n=1}^{N} \left[\frac{\exp\left(\mathbf{w}_i^T \mathbf{x}_n\right)}{\sum_{l=1}^{K} \exp\left(\mathbf{w}_l^T \mathbf{x}_n\right)} - y_{in} \right] \mathbf{x}_n \quad \forall\, i \in [1, k], \tag{4.9}$$

which can be written as matrix products $\boldsymbol{\alpha_i} = \mathbf{X}^T \mathbf{w}_i$ and $\nabla f_i(\mathbf{W}) = \mathbf{X}\boldsymbol{\beta}_i$, where the entries of $\boldsymbol{\beta}_i \in \mathbb{R}^N$ are given by $\beta_{in} = \left(\frac{\exp(\alpha_{in})}{\sum_{l=1}^{K} \exp(\alpha_{ln})} - y_{in} \right)$. Thus, the full gradient matrix is given by $\nabla f(\mathbf{W}) = \mathbf{X}\boldsymbol{\beta}$ where the entries of $\boldsymbol{\beta} \in \mathbb{R}^{N \times K}$ are dependent on $\boldsymbol{\alpha} \in \mathbb{R}^{N \times K}$ as above and the matrix $\boldsymbol{\alpha}$ is given by $\boldsymbol{\alpha} = \mathbf{X}^T \mathbf{W}$. We assume that the number of classes K is small enough such that tall matrices $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are small enough for the master to do local calculations on them.

Since the effective number of features is $d \times K$, the Hessian matrix is of dimension $dK \times dK$. The (i, j)-th component of the Hessian, say $\mathbf{H}_{ij}$, is

$$\mathbf{H}_{ij}(\mathbf{W}) = \frac{d}{d\mathbf{w}_j} \nabla f_i(\mathbf{W}) = \frac{d}{d\mathbf{w}_j} \mathbf{X}\boldsymbol{\beta_i} = \mathbf{X} \frac{d}{d\mathbf{w}_j} \boldsymbol{\beta}_i = \mathbf{X}\mathbf{Z_{ij}}\mathbf{X}^T \tag{4.10}$$

where $\mathbf{Z}_{ij} \in \mathbb{R}^{N \times N}$ is a diagonal matrix whose n-th diagonal entry is

$$Z_{ij}(n) = \frac{\exp(\alpha_{in})}{\sum_{l=1}^{K} \exp(\alpha_{ln})} \left(\mathbb{I}(i = j) - \frac{\exp(\alpha_{jn})}{\sum_{l=1}^{K} \exp(\alpha_{ln})} \right) \ \forall\, n \in [1, N], \tag{4.11}$$

where $\mathbb{I}(\cdot)$ is the indicator function and $\boldsymbol{\alpha} = \mathbf{XW}$ was defined above. The full Hessian matrix is obtained by putting together all such $\mathbf{H}_{ij}$'s in a $dK \times dK$ matrix and can be expressed in a matrix-matrix multiplication form as

$$\nabla^2 f(\mathbf{W}) = \begin{bmatrix} \mathbf{H}_{11} & \cdots & \mathbf{H}_{1K} \\ \vdots & \ddots & \vdots \\ \mathbf{H}_{K1} & \cdots & \mathbf{H}_{KK} \end{bmatrix} = \begin{bmatrix} \mathbf{X}\mathbf{Z}_{11}\mathbf{X}^T & \cdots & \mathbf{X}\mathbf{Z}_{1K}\mathbf{X}^T \\ \vdots & \ddots & \vdots \\ \mathbf{X}\mathbf{Z}_{K1}\mathbf{X}^T & \cdots & \mathbf{X}\mathbf{Z}_{KK}\mathbf{X}^T \end{bmatrix} = \bar{\mathbf{X}}\bar{\mathbf{Z}}\bar{\mathbf{X}}^T, \tag{4.12}$$

[6] We also evaluate the exact Hessian-based algorithm with speculative execution, i.e., recomputing the straggling jobs, and compare it with OverSketched Newton in Sec. 4.5.

where $\bar{\mathbf{X}} \in \mathbb{R}^{dK \times NK}$ is a block diagonal matrix that contains $\mathbf{X}$ in the diagonal blocks and $\bar{\mathbf{Z}} \in \mathbb{R}^{NK \times NK}$ is formed by stacking all the $\mathbf{Z}_{ij}$'s for $i, j \in [1, K]$. In OverSketched Newton, we compute this multiplication using sketching in serverless systems for efficiency and resiliency to stragglers. Assuming $d \times K$ is small enough, the master can then calculate the update $\mathbf{p}_t$ using efficient algorithms such the minimum-residual method [143], [153].

4.4.3 Other Example Problems

In this section, we describe several other commonly encountered optimization problems that can be solved using OverSketched Newton.

Ridge Regularized Linear Regression: The optimization problem is

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2n} ||\mathbf{X}^T \mathbf{w} - \mathbf{y}||_2^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2. \tag{4.13}$$

The gradient in this case can be written as $\frac{1}{n}\mathbf{X}(\boldsymbol{\beta} - \mathbf{y}) + \lambda \mathbf{w}$, where $\boldsymbol{\beta} = \mathbf{X}^T \mathbf{w}$, where the training matrix $\mathbf{X}$ and label vector $\mathbf{y}$ were defined previously. The Hessian is given by $\nabla^2 f(\mathbf{w}) = \mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}$. For $n \gg d$, this can be computed approximately using the sketch matrix in (4.4).

Linear programming via interior point methods: The following linear program can be solved using OverSketched Newton

$$\underset{\mathbf{A}\mathbf{x} \leq \mathbf{b}}{\text{minimize}}\ \mathbf{c}^T \mathbf{x}, \tag{4.14}$$

where $\mathbf{x} \in \mathbb{R}^{m \times 1}, \mathbf{c} \in \mathbb{R}^{m \times 1}, \mathbf{b} \in \mathbb{R}^{n \times 1}$ and $\mathbf{A} \in \mathbb{R}^{n \times m}$ is the constraint matrix with $n > m$. In algorithms based on interior point methods, the following sequence of problems using Newton's method

$$\min_{\mathbf{x} \in \mathbb{R}^m} f(\mathbf{x}) = \min_{\mathbf{x} \in \mathbb{R}^m} \left(\tau \mathbf{c}^T \mathbf{x} - \sum_{i=1}^{n} \log(b_i - \mathbf{a}_i \mathbf{x}) \right), \tag{4.15}$$

where $\mathbf{a}_i$ is the i-th row of $\mathbf{A}$, τ is increased geometrically such that when τ is very large, the logarithmic term does not affect the objective value and serves its purpose of keeping all intermediates solution inside the constraint region. The update in the t-th iteration is given by $\mathbf{x}_{t+1} = \mathbf{x}_t - (\nabla^2 f(\mathbf{x}_t))^{-1} \nabla f(\mathbf{x}_t)$, where $\mathbf{x}_t$ is the estimate of the solution in the t-th iteration. The gradient can be written as $\nabla f(\mathbf{x}) = \tau \mathbf{c} + \mathbf{A}^T \boldsymbol{\beta}$ where $\beta_i = 1/(b_i - \alpha_i)$ and $\boldsymbol{\alpha} = \mathbf{A}\mathbf{x}$.

The Hessian for the objective in (4.15) is given by

$$\nabla^2 f(\mathbf{x}) = \mathbf{A}^T \text{diag} \frac{1}{(b_i - \alpha_i)^2} \mathbf{A}. \tag{4.16}$$

The square root of the Hessian is given by $\nabla^2 f(\mathbf{x})^{1/2} = \text{diag} \frac{1}{|bi - \alpha_i|} \mathbf{A}$. The computation of Hessian requires $O(nm^2)$ time and is the bottleneck in each iteration. Thus, we can use sketching to mitigate stragglers while evaluating the Hessian efficiently, i.e. $\nabla^2 f(\mathbf{x}) \approx (\mathbf{S} \nabla^2 f(\mathbf{x})^{1/2})^T \times (\mathbf{S} \nabla^2 f(\mathbf{x})^{1/2})$, where $\mathbf{S}$ is the OverSketch matrix defined in (4.4).

Lasso Regularized Linear Regression: The optimization problem takes the following form

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}||\mathbf{X}\mathbf{w}-\mathbf{y}||_2^2 + \lambda||\mathbf{w}||_1, \tag{4.17}$$

where $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the measurement matrix, the vector $\mathbf{y} \in \mathbb{R}^n$ contains the measurements, $\lambda \geq 0$ and $d \gg n$. To solve (4.17), we consider its dual variation

$$\min_{||\mathbf{X}^T\mathbf{z}||_\infty \leq \lambda, \mathbf{z}\in\mathbb{R}^n} \frac{1}{2}||\mathbf{y}-\mathbf{z}||_2^2,$$

which is amenable to interior point methods and can be solved by optimizing the following sequence of problems where τ is increased geometrically

$$\min_{\mathbf{z}} f(\mathbf{z}) = \min_{\mathbf{z}} \Big(\frac{\tau}{2}||\mathbf{y}-\mathbf{z}||_2^2 - \sum_{j=1}^{d}\log(\lambda-\mathbf{x}_j^T\mathbf{z}) - \sum_{j=1}^{d}(\lambda+\mathbf{x}_j^T\mathbf{z})\Big),$$

where $\mathbf{x}_j$ is the j-th column of $\mathbf{X}$. The gradient can be expressed in few matrix-vector multiplications as $\nabla f(\mathbf{z}) = \tau(\mathbf{z}-\mathbf{y}) + \mathbf{X}(\boldsymbol{\beta}-\boldsymbol{\gamma})$, where $\beta_i = 1/(\lambda-\alpha_i)$, $\gamma_i = 1/(\lambda+\alpha_i)$, and $\boldsymbol{\alpha} = \mathbf{X}^T\mathbf{z}$. Similarly, the Hessian can be written as $\nabla^2 f(\mathbf{z}) = \tau\mathbf{I} + \mathbf{X}\boldsymbol{\Lambda}\mathbf{X}^T$, where $\boldsymbol{\Lambda}$ is a diagonal matrix whose entries are given by $\Lambda_{ii} = 1/(\lambda-\alpha_i)^2 + 1/(\lambda+\alpha_i)^2 \; \forall \; i \in [1, n]$.

Other common problems where OverSketched Newton is applicable include Linear Regression, Support Vector Machines (SVMs), Semidefinite programs, etc.

4.5 Experimental Results

In this section, we evaluate OverSketched Newton on AWS Lambda using real-world and synthetic datasets, and we compare it with state-of-the-art distributed optimization algorithms[7]. We use the serverless computing framework, Pywren [1]. Our experiments are focused on logistic and softmax regression, which are popular supervised learning problems, but they can be reproduced for other problems described in Section 4.4. We present experiments on the following datasets:

Dataset	Training Samples	Features	Testing samples
Synthetic	$300,000$	3000	$100,000$
EPSILON	$400,000$	2000	$100,000$
WEBPAGE	$48,000$	300	$15,000$
a9a	$32,000$	123	$16,000$
EMNIST	$240,000$	7840	$40,000$

[7] A working implementation of OverSketched Newton is available at https://github.com/vvipgupta/OverSketchedNewton

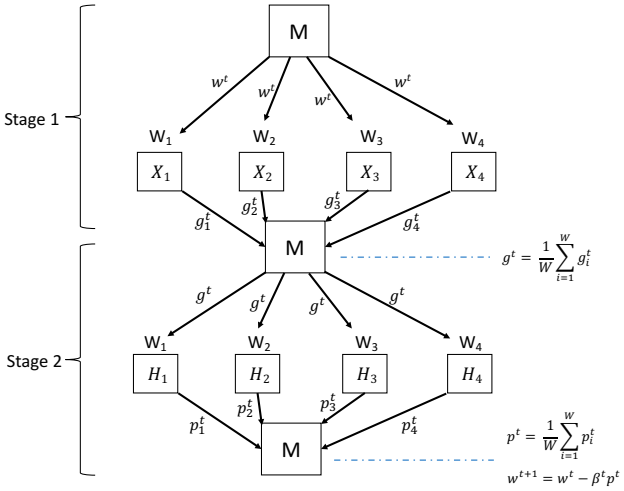
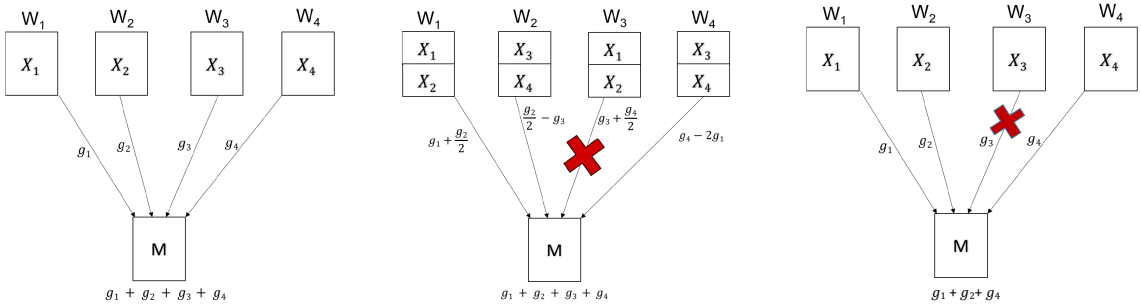


Figure 4.3: GIANT: The two stage second order distributed optimization scheme with four workers. First, master calculates the full gradient by aggregating local gradients from workers. Second, the master calculates approximate Hessian using local second-order updates from workers.



(a) Simple Gradient Descent where each worker stores one-fourth fraction of the whole data and sends back a partial gradient corresponding to its own data to the master

(b) Gradient Coding described in [25] with W_3 straggling. To get the global gradient, master would compute $g_1 + g_2 + g_3 + g_4 = 3\left(g_1 + \frac{g_2}{2}\right) - \left(\frac{g2}{2} - g_3\right) + (g_4 - 2g_1)$

(c) Mini-batch gradient descent, where the stragglers are ignored during gradient aggregation and the gradient is later scaled according to the size of mini-batch

Figure 4.4: Different gradient descent schemes in serverful systems in presence of stragglers

For comparison of OverSketched Newton with existing distributed optimization schemes, we choose recently-proposed Globally Improved Approximate Newton Direction (GIANT) [51]. The reason is that GIANT boasts a better convergence rate than many existing distributed second-order methods for linear and logistic regression, when $n \gg d$. In GIANT, and other similar distributed second-order algorithms, the training data is evenly divided among workers, and the algorithms proceed in two stages. First, the workers compute partial gradients using local training data, which is then aggregated by the master to compute the exact gradient. Second, the workers receive the full

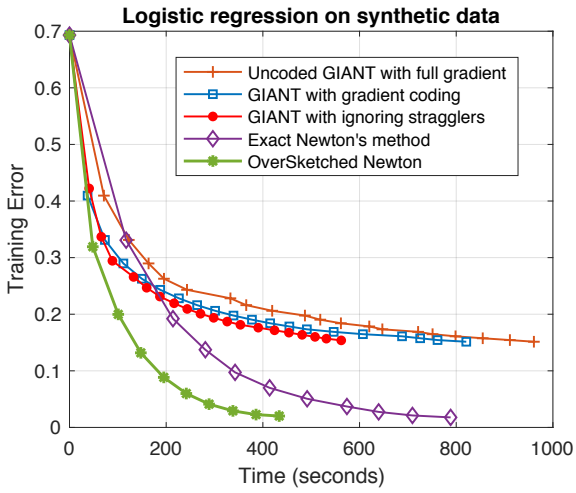


Figure 4.5: Convergence comparison of GIANT (employed with different straggler mitigation methods), exact Newton's method and OverSketched Newton for Logistic regression on AWS Lambda. The synthetic dataset considered has 300,000 examples and 3000 features.

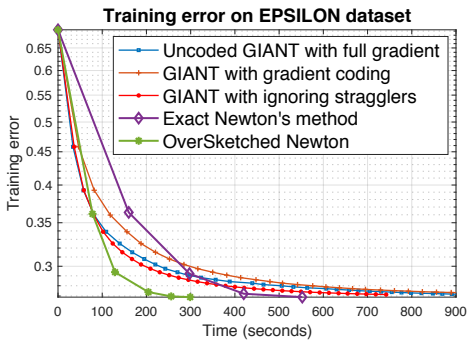
gradient to calculate their local second-order estimate, which is then averaged by the master. An illustration is shown in Fig. 4.3.

For straggler mitigation in such serverful systems based algorithms, [25] proposes a scheme for coding gradient updates called *gradient coding*, where the data at each worker is repeated multiple times to compute redundant copies of the gradient. See Figure 4.4b for illustration. Figure 4.4a illustrates the scheme that waits for all workers and Figure 4.4c illustrates the ignoring stragglers approach. We use the three schemes for dealing with stragglers illustrated in Figure 4.4 during the two stages of GIANT, and we compare their convergence with OverSketched Newton. We further evaluate and compare the convergence exact Newton's method (employed with speculative execution, that is, reassigning and recomputing the work for straggling workers).

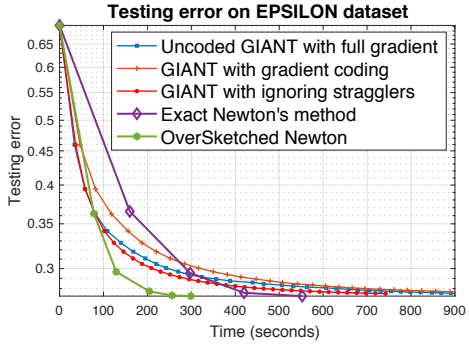
4.5.1 Comparisons with Existing Second-Order Methods on AWS Lambda

In Figure 4.5, we present our results on a synthetic dataset with $n = 300,000$ and $d = 3000$ for logistic regression on AWS Lambda. Each column $\mathbf{x}_i \in \mathbb{R}^d$, for all $i \in [1, n]$, is sampled uniformly randomly from the cube $[-1, 1]^d$. The labels y_i are sampled from the logistic model, that is, $\mathbb{P}[y_i = 1] = 1/(1 + exp(\mathbf{x}_i\mathbf{w} + b))$, where the weight vector $\mathbf{w}$ and bias b are generated randomly from the normal distribution. ht vector $\mathbf{w}$ and bias b are generated randomly from the normal distribution.

The orange, blue and red curves demonstrate the convergence for GIANT with the full gradient (that waits for all the workers), gradient coding and mini-batch gradient (that ignores the stragglers while calculating gradient and second-order updates) schemes, respectively. The purple and green curves depict the convergence for the exact Newton's method and OverSketched Newton, respectively. The gradient coding scheme is applied for one straggler, that is the data is repeated



(a) Training error for logistic regression on EPSILON dataset



(b) Testing error for logistic regression on EPSILON dataset

Figure 4.6: Comparison of training and testing errors for logistic regression on EPSILON dataset with several Newton based schemes on AWS Lambda. OverSketched Newton outperforms others by at least 46%. Testing error closely follows training error.

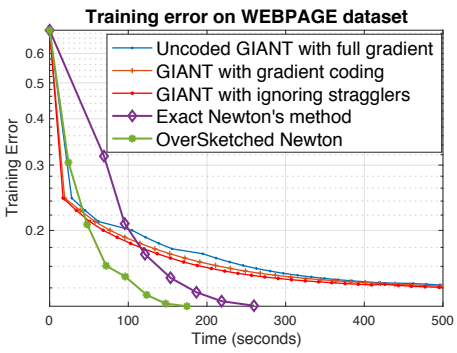
twice at each worker. We use 60 Lambda workers for executing GIANT in parallel. Similarly, for Newton's method, we use 60 workers for matrix-vector multiplication in steps 4 and 8 of Algorithm 11, 3600 workers for exact Hessian computation and 600 workers for sketched Hessian computation with a sketch dimension of $10d = 30,000$ in step 14 of Algorithm 11. In all cases, unit step-size was used to update the model[8]

Remark 5. In our experiments, we choose the number of workers in such a way that each worker receives approximately the same amount of data to work with, regardless of the algorithm. This is motivated by the fact that the memory at each worker is the bottleneck in serverless systems (e.g., in AWS Lambda, the memory at each worker can be as low as 128 MB). Note that this is unlike serverful/HPC systems, where the number of workers is the bottleneck.

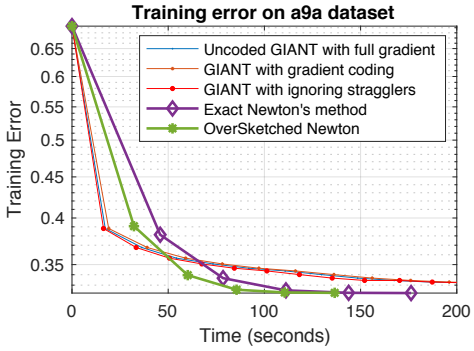
An important point to note from Fig. 4.5 is that the uncoded scheme (that is, the one that waits for all stragglers) has the worst performance. The implication is that good straggler/fault mitigation algorithms are essential for computing in the serverless setting. Secondly, the mini-batch scheme outperforms the gradient coding scheme by 25%. This is because gradient coding requires additional communication of data to serverless workers (twice when coding for one straggler, see [25] for details) at each invocation to AWS Lambda. On the other hand, the exact Newton's method converges much faster than GIANT, even though it requires more time per iteration.

The number of iterations needed for convergence for OverSketched Newton and exact Newton (that exactly computes the Hessian) is similar, but OverSketched Newton converges in almost half the time due to an efficient computation of (approximate) Hessian (which is the computational bottleneck and thus reduces time per iteration).

[8]Line-search in Section 4.3 was mainly introduced to prove theoretical guarantees. In our experiments, we observe that constant step-size works well for OverSketched Newton.



(a) Logistic regression on WEBPAGE dataset



(b) Logistic regression on a9a dataset

Figure 4.7: Logistic regression on WEBPAGE and a9a datasets with several Newton based schemes on AWS Lambda. OverSketched Newton outperforms others by at least 25%.

Logistic Regression on EPSILON, WEBPAGE and a9a Datasets

In Figure 4.6, we repeat the above experiment with EPSILON classification dataset obtained from [119], with $n = 0.4\,million$ and $d = 2000$. We plot training and testing errors for logistic regression for the schemes described in the previous section. Here, we use 100 workers for GIANT, and 100 workers for matrix-vector multiplications for gradient calculation in OverSketched Newton. We use gradient coding designed for three stragglers in GIANT. This scheme performs worse than uncoded GIANT that waits for all the stragglers due to the repetition of training data at workers. Hence, one can conclude that the communication costs dominate the straggling costs. In fact, it can be observed that the mini-batch gradient scheme that ignores the stragglers outperforms the gradient coding and uncoded schemes for GIANT.

During exact Hessian computation, we use $10,000$ serverless workers with speculative execution to mitigate stragglers (i.e., recomputing the straggling jobs) compared to OverSketched Newton that uses 1500 workers with a sketch dimension of $15d = 30,000$. OverSketched Newton requires a significantly smaller number of workers, as once the square root of Hessian is sketched in a distributed fashion, it can be copied into local memory of the master due to dimension reduction, and the Hessian can be calculated locally. Testing error follows training error closely, and important conclusions remain the same as in Figure 4.5. OverSketched Newton outperforms GIANT and exact Newton-based optimization by at least 46% in terms of running time.

We repeated the above experiments for classification on the WEBPAGE ($n = 49,749$ and $d = 300$) and a9a ($n = 32,561$ and $d = 123$) datasets [119]. For both datasets, we used 30 workers for each iteration in GIANT and any matrix-vector multiplications. Exact hessian calculation invokes 900 workers as opposed to 300 workers for OverSketched Newton, where the sketch dimension was $10d = 3000$. The results for training loss on logistic regression are shown in Figure 4.7. Testing error closely follows the training error in both cases. OverSketched Newton outperforms exact Newton and GIANT by at least $\sim 25\%$ and $\sim 75\%$, respectively, which is similar to the trends witnessed heretofore.

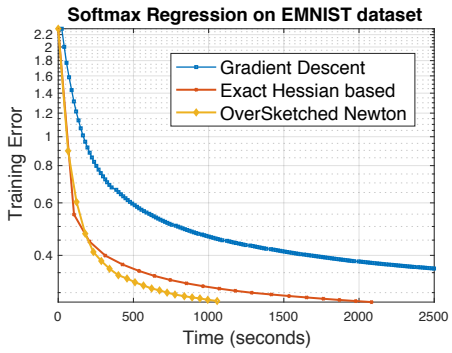


Figure 4.8: Convergence comparison of gradient descent, exact Newton's method and OverSketched Newton for Softmax regression on AWS Lambda.

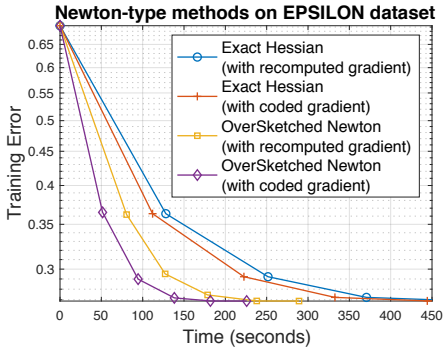


Figure 4.9: Convergence comparison of speculative execution and coded computing for gradient and Hessian computing with logistic regression on AWS Lambda.

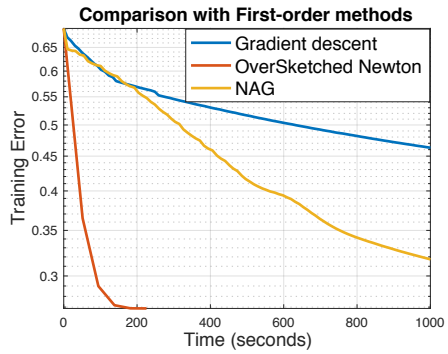


Figure 4.10: Convergence comparison of gradient descent, NAG and OverSketched Newton on AWS Lambda.

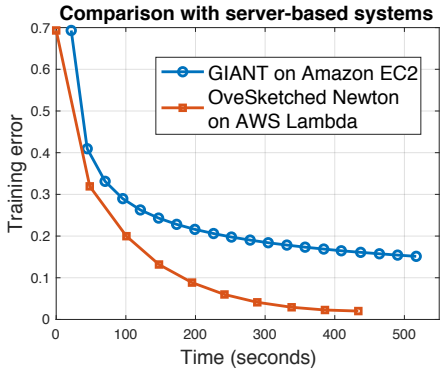


Figure 4.11: Convergence comparison of GIANT on AWS EC2 and OverSketched Newton on AWS Lambda.

Remark 6. Note that conventional distributed second-order methods for serverful systems—which distribute training examples evenly across workers (such as [48]–[53])—typically find a "localized approximation" (localized to each machine) of second-order update at each worker and then aggregate it. OverSketched Newton, on the other hand, uses the massive storage and compute power in serverless systems to find a more "globalized approximation" (globalized in the sense of across machine). Thus, it performs better in practice.

4.5.2 Softmax Regression on EMIST

In Fig. 4.8, we solve unregularized softmax regression, which is weakly convex (see Sec. 4.4.2 for details). We use the Extended MNIST (EMNIST) dataset [154] with $N = 240,000$ training examples, $d = 784$ features and $K = 10$ classes. Note that GIANT cannot be applied here as the objective function is not strongly convex. We compare the convergence rate of OverSketched

Newton, exact Hessian and gradient descent based schemes.

For gradient computation in all three schemes, we use 60 workers. However, exact Newton scheme requires 3600 workers to calculate the $dK \times dK$ Hessian and recomputes the straggling jobs, while OverSketched Newton requires only 360 workers to calculate the sketch in parallel with sketch dimension $6dK = 47,040$. The approximate Hessian is then computed locally at the master using its sketched square root, where the sketch dimension is $6dK = 47,040$. The step-size is fixed and is determined by hyperparamter tuning before the start of the algorithm. Even for the weakly-convex case, second-order methods tend to perform better. Moreover, the runtime of OverSketched Newton outperforms both gradient descent and Exact Newton based methods by $\sim 75\%$ and $\sim 50\%$, respectively.

4.5.3 Coded computing versus Speculative Execution

In Figure 4.9, we compare the effect of straggler mitigation schemes, namely speculative execution, that is, restarting the jobs with straggling workers, and coded computing on the convergence rate during training and testing. We regard OverSketch based matrix multiplication as a coding scheme in which some redundancy is introduced during "over" sketching for matrix multiplication. There are four different cases, corresponding to gradient and hessian calculation using either speculative execution or coded computing. For speculative execution, we wait for at least 90% of the workers to return (this works well as the number of stragglers is generally less than 10%) and restart the jobs that did not return till this point.

For both exact Hessian and OverSketched Newton, using codes for distributed gradient computation outperforms speculative execution based straggler mitigation. Moreover, computing the Hessian using OverSketch is significantly better than exact computation in terms of running time as calculating the Hessian is the computational bottleneck in each iteration.

4.5.4 Comparison with First-Order Methods on AWS Lambda

In Figure 4.10, we compare gradient descent and Nesterov Accelerated Gradient (NAG) (while ignoring the stragglers) with OverSketched Newton for logistic regression on EPSILON dataset. We observed that for first-order methods, there is only a slight difference in convergence for a mini-batch gradient when the batch size is 95%. Hence, for gradient descent and NAG, we use 100 workers in each iteration while ignoring the stragglers.[9] These first-order methods were given the additional advantage of backtracking line-search, which determined the optimal amount to move in given a descent direction.[10] Overall, OverSketched Newton with unit step-size significantly outperforms gradient descent and NAG with backtracking line-search.

[9] We note that stochastic methods such as SGD perform worse that gradient descent since their update quality is poor, requiring more iterations (hence, more communication) to converge while not using the massive compute power of serverless. For example, 20% minibatch SGD in the setup of Fig. 4.10 requires $1.9\times$ more time than gradient descent with same number of workers.

[10] We remark that backtracking line-search required $\sim 13\%$ of the total time for NAG. Hence, as can be seen from Fig. 4.10, any well-tuned step-size method would still be significantly slower than OverSketched Newton.

4.5.5 Comparison with Serverful Optimization

In Fig. 4.11, we compare OverSketched Newton on AWS Lambda with existing distributed optimization algorithm GIANT in serverful systems (AWS EC2). The results are plotted on synthetically generated data for logistic regression. For serverful programming, we use Message Passing Interface (MPI) with one `c3.8xlarge` master and 60 `t2.medium` workers in AWS EC2. In [3], the authors observed that many large-scale linear algebra operations on serverless systems take at least 30% more time compared to MPI-based computation on serverful systems. However, as shown in Fig. 4.11, we observe a slightly surprising trend that OverSketched Newton outperforms MPI-based optimization (that uses existing state-of-the-art optimization algorithm). This is because OverSketched Newton exploits the flexibility and massive scale at disposal in serverless, and thus produces a better approximation of the second-order update than GIANT.[11]

4.6 Proofs

To complete the proofs in this section, we will need the following lemma.

Lemma 4. *Let $\hat{\mathbf{H}}_t = \mathbf{A}_t^T \mathbf{S}_t \mathbf{S}_t^T \mathbf{A}_t$ where $\mathbf{S}_t$ is the sparse sketch matrix in* (3.3) *with sketch dimension $m = \Omega(d^{1+\mu}/\epsilon^2)$ and $N = \Theta_\mu(1/\epsilon)$. Then, the following holds*

$$\lambda_{\min}(\hat{\mathbf{H}}_t) \geq (1-\epsilon)\lambda_{\min}(\nabla^2 f(\mathbf{w}_t)), \tag{4.18}$$

$$\lambda_{\max}(\hat{\mathbf{H}}_t) \leq (1+\epsilon)\lambda_{\max}(\nabla^2 f(\mathbf{w}_t)) \tag{4.19}$$

with probability at least $1 - \frac{1}{d^\tau}$, where $\tau > 0$ is a constant depending on μ and the constants in $\Theta(\cdot)$ and $\Omega(\cdot)$, and $\lambda_{\max}(\cdot)$ and $\lambda_{\min}(\cdot)$ denote the maximum and minimum eigenvalues, respectively. In general,

$$\lambda_i(\nabla^2 f(\mathbf{w}_t)) - \epsilon\lambda_{\max}(\nabla^2 f(\mathbf{w}_t)) \leq \lambda_i(\hat{\mathbf{H}}_t) \leq \lambda_i(\nabla^2 f(\mathbf{w}_t)) + \epsilon\lambda_{\max}(\nabla^2 f(\mathbf{w}_t)),$$

where $\lambda_i(\cdot)$ is the i-th eigenvalue.

Proof. We note than N is the number of non-zero elements per row in the sketch $\mathbf{S}_t$ in (3.3) after ignoring stragglers. We use Theorem 8 in [155] to bound the singular values for the sparse sketch $\mathbf{S}_t$ in (3.3) with sketch dimension $m = \Omega(d^{1+\mu}/\epsilon^2)$ and $N = \Theta(1/\epsilon)$. It says that $\mathbb{P}(\forall\ \mathbf{x} \in \mathbb{R}^n, ||\mathbf{S}_t\mathbf{x}||_2 \in (1 \pm \epsilon/3)||\mathbf{x}||_2) > 1 - 1/d^\tau$, where $\tau > 0$ depends on μ and the constants in $\Theta(\cdot)$ and $\Omega(\cdot)$. Thus, $||\mathbf{S}_t\mathbf{x}||_2 \in (1 \pm \epsilon/3)||\mathbf{x}||_2$, which implies that

$$||\mathbf{S}_t\mathbf{x}||_2^2 \in (1 + \epsilon^2/9 \pm 2\epsilon/3)||\mathbf{x}||_2^2,$$

[11] We do not compare with exact Newton in serverful sytems since the data is large and stored in the cloud. Computing the exact Hessian would require a large number of workers (e.g., we use 10,000 workers for exact Newton in EPSILON dataset) which is infeasible in existing serverful systems.

with probability at least $1 - 1/d^\tau$. For $\epsilon \leq 1/2$, this leads to the following inequality

$$||\mathbf{S}_t\mathbf{x}||_2^2 \in (1 \pm \epsilon)||\mathbf{x}||_2^2 \Rightarrow |\mathbf{x}^T(\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I})\mathbf{x}| \leq \epsilon||\mathbf{x}||_2^2 \; \forall \; x \in \mathbb{R}^n \tag{4.20}$$

with probability at least $1 - 1/d^\tau$. Also, since $(1-\epsilon)\mathbf{x}^T\mathbf{x} \leq \mathbf{x}^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{x} \leq (1+\epsilon)\mathbf{x}^T\mathbf{x} \; \forall \; x \in \mathbb{R}^n$ by the inequality above, replacing $\mathbf{x}$ by $\mathbf{A}\mathbf{y}$, we get

$$(1-\epsilon)\mathbf{y}^T\mathbf{A}^T\mathbf{A}\mathbf{y} \leq \mathbf{y}^T\mathbf{A}^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{A}\mathbf{y} \leq (1+\epsilon)\mathbf{y}^T\mathbf{A}^T\mathbf{A}\mathbf{y} \tag{4.21}$$

with probability at least $1-1/d^\tau$. Let $\mathbf{y}_1$ be the unit norm eigenvector corresponding to the minimum eigenvalue of $\hat{\mathbf{H}}_t = \mathbf{A}_t^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{A}_t$. Since the above inequality is true for all $\mathbf{y}$, we have

$$\begin{aligned}\mathbf{y}_1^T\mathbf{A}_t^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{A}_t\mathbf{y}_1 \geq (1-\epsilon)\mathbf{y}_1^T\mathbf{A}_t^T\mathbf{A}_t\mathbf{y}_1 \geq (1-\epsilon)\lambda_{\min}(\mathbf{A}_t^T\mathbf{A}_t) = (1-\epsilon)\lambda_{\min}(\nabla^2 f(\mathbf{w}_t)) \\ \Rightarrow \lambda_{\min}(\hat{\mathbf{H}}_t) \geq (1-\epsilon)\lambda_{\min}(\nabla^2 f(\mathbf{w}_t))\end{aligned}$$

with probability at least $1 - 1/d^\tau$. Along similar lines, we can prove that $\lambda_{\max}(\hat{\mathbf{H}}_t) \leq (1-\epsilon)\lambda_{\max}(\nabla^2 f(\mathbf{w}_t))$ with probability at least $1 - 1/d^\tau$ using the right hand inequality in (4.21). Together, these prove the first result.

In general, Eq. (4.20) implies that the eigenvalues of $(\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I})$ are in the set $[-\epsilon, \epsilon]$. Thus, all the eigenvalues of $\mathbf{A}_t^T(\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I})\mathbf{A}_t$ are in the set $[-\epsilon\lambda_{\max}(\nabla^2 f(\mathbf{w}_t)), \epsilon\lambda_{\max}(\nabla^2 f(\mathbf{w}_t))]$ Also, we can write

$$\hat{\mathbf{H}}_t = \mathbf{A}_t^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{A}_t = \mathbf{A}_t^T\mathbf{A}_t + \mathbf{A}_t^T(\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I})\mathbf{A}_t.$$

Now, applying Weyl's inequality (see [156], Section 1.3) on symmetric matrices $\hat{\mathbf{H}}_t = \mathbf{A}_t^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{A}_t$, $\nabla^2 f(\mathbf{w}_t) = \mathbf{A}_t^T\mathbf{A}_t$ and $\mathbf{A}_t^T(\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I})\mathbf{A}_t$, we get

$$\lambda_i(\nabla^2 f(\mathbf{w}_t)) - \epsilon\lambda_{\max}(\nabla^2 f(\mathbf{w}_t)) \leq \lambda_i(\hat{\mathbf{H}}_t) \leq \lambda_i(\nabla^2 f(\mathbf{w}_t)) + \epsilon\lambda_{\max}(\nabla^2 f(\mathbf{w}_t)),$$

which proves the second result.

□

4.6.1 Proof of Theorem 5

Let's define $\mathbf{w}_\tau = \mathbf{w}_t + \tau\mathbf{p}_t$, where the descent direction $\mathbf{p}_t$ is given by $\mathbf{p}_t = -\hat{\mathbf{H}}_t^{-1}\nabla f(\mathbf{w}_t)$. Also, from Lemma 4, we have

$$\lambda_{\min}(\hat{\mathbf{H}}_t) \geq (1-\epsilon)\lambda_{\min}(\nabla^2 f(\mathbf{w}_t)) \text{ and } \lambda_{\max}(\hat{\mathbf{H}}_t) \leq (1+\epsilon)\lambda_{\max}(\nabla^2 f(\mathbf{w}_t)),$$

with probability at least $1 - 1/d^\tau$. Using the above inequalities and the fact that $f(\cdot)$ is k-strongly convex and M-smooth, we get

$$(1-\epsilon)k\mathbf{I} \preceq \hat{\mathbf{H}}_t \preceq (1+\epsilon)M\mathbf{I}, \tag{4.22}$$

with probability at least $1 - 1/d^\tau$.

Next, we show that there exists an $\alpha > 0$ such that the Armijo line search condition in (4.5) is satisfied. From the smoothness of $f(\cdot)$, we get (see [157], Theorem 2.1.5)

$$\begin{aligned} f(\mathbf{w}_\alpha) - f(\mathbf{w}_t) &\leq (\mathbf{w}_\alpha - \mathbf{w}_t)^T \nabla f(\mathbf{w}_t) + \frac{M}{2}||\mathbf{w}_\alpha - \mathbf{w}_t||^2, \\ &= \alpha \mathbf{p}_t^T \nabla f(\mathbf{w}_t) + \alpha^2 \frac{M}{2}||\mathbf{p}_t||^2. \end{aligned}$$

Now, for $\mathbf{w}_\alpha$ to satisfy the Armijo rule, α should satisfy

$$\begin{aligned} \alpha \mathbf{p}_t^T \nabla f(\mathbf{w}_t) + \alpha^2 \frac{M}{2}||\mathbf{p}_t||^2 &\leq \alpha\beta \mathbf{p}_t^T \nabla f(\mathbf{w}_t) \\ \Rightarrow \alpha \frac{M}{2}||\mathbf{p}_t||^2 &\leq (\beta - 1)\mathbf{p}_t^T \nabla f(\mathbf{w}_t) \\ \Rightarrow \alpha \frac{M}{2}||\mathbf{p}_t||^2 &\leq (1 - \beta)\mathbf{p}_t^T \hat{\mathbf{H}}_t \mathbf{p}_t, \end{aligned}$$

where the last inequality follows from the definition of $\mathbf{p}_t$. Now, using the lower bound from (4.22), $\mathbf{w}_\alpha$ satisfies Armijo rule for all

$$\alpha \leq \frac{2(1-\beta)(1-\epsilon)k}{M}.$$

Hence, we can always find an $\alpha_t \geq \frac{2(1-\beta)(1-\epsilon)k}{M}$ using backtracking line search such that $\mathbf{w}_{t+1}$ satisfies the Armijo condition, that is,

$$\begin{aligned} f(\mathbf{w}_{t+1}) - f(\mathbf{w}_t) &\leq \alpha_t \beta \mathbf{p}_t^T \nabla f(\mathbf{w}_t) \\ &= -\alpha_t \beta \nabla f(\mathbf{w}_t)^T \hat{\mathbf{H}}_t^{-1} \nabla f(\mathbf{w}_t) \\ &\leq -\frac{\alpha_t \beta}{\lambda_{\max}(\hat{\mathbf{H}}_t)}||\nabla f(\mathbf{w}_t)||^2 \end{aligned}$$

which in turn implies

$$f(\mathbf{w}_t) - f(\mathbf{w}_{t+1}) \geq \frac{\alpha_t \beta}{M(1+\epsilon)}||\nabla f(\mathbf{w}_t)||^2 \tag{4.23}$$

with probability at least $1 - 1/d^\tau$. Here the last inequality follows from the bound in (4.22). Moreover, k-strong convexity of $f(\cdot)$ implies (see [157], Theorem 2.1.10)

$$f(\mathbf{w}_t) - f(\mathbf{w}^*) \leq \frac{1}{2k}||\nabla f(\mathbf{w}_t)||^2.$$

Using the inequality from (4.23) in the above inequality, we get

$$\begin{aligned} f(\mathbf{w}_t) - f(\mathbf{w}_{t+1}) &\geq \frac{2\alpha_t \beta k}{M(1+\epsilon)}(f(\mathbf{w}_t) - f(\mathbf{w}^*)) \\ &\geq \rho(f(\mathbf{w}_t) - f(\mathbf{w}^*)), \end{aligned}$$

where $\rho = \frac{2\alpha_t \beta k}{M(1+\epsilon)}$. Rearranging, we get

$$f(\mathbf{w}_{t+1}) - f(\mathbf{w}^*) \leq (1-\rho)(f(\mathbf{w}_t) - f(\mathbf{w}^*))$$

with probability at least $1 - 1/d^\tau$, which proves the desired result.

4.6.2 Proof of Theorem 6

According to OverSketched Newton update, $\mathbf{w}_{t+1}$ is obtained by solving

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \Big\{ f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^T(\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^T \hat{\mathbf{H}}_t (\mathbf{w} - \mathbf{w}_t) \Big\}.$$

Thus, we have, for any $\mathbf{w} \in \mathbb{R}^d$,

$$\begin{aligned}
&f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^T(\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^T \hat{\mathbf{H}}_t (\mathbf{w} - \mathbf{w}_t), \\
&\quad \geq f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^T(\mathbf{w}_{t+1} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w}_{t+1} - \mathbf{w}_t)^T \hat{\mathbf{H}} (\mathbf{w}_{t+1} - \mathbf{w}_t), \\
&\Rightarrow \nabla f(\mathbf{w}_t)^T(\mathbf{w} - \mathbf{w}_{t+1}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^T \hat{\mathbf{H}}_t (\mathbf{w} - \mathbf{w}_t) - \frac{1}{2}(\mathbf{w}_{t+1} - \mathbf{w}_t)^T \hat{\mathbf{H}}_t (\mathbf{w}_{t+1} - \mathbf{w}_t) \geq 0, \\
&\Rightarrow \nabla f(\mathbf{w}_t)^T(\mathbf{w} - \mathbf{w}_{t+1}) + \frac{1}{2}\Big[(\mathbf{w} - \mathbf{w}_t)^T \hat{\mathbf{H}}_t (\mathbf{w} - \mathbf{w}_{t+1}) + (\mathbf{w} - \mathbf{w}_{t+1})^T \hat{\mathbf{H}}_t (\mathbf{w}_{t+1} - \mathbf{w}_t)\Big] \geq 0.
\end{aligned}$$

Substituting $\mathbf{w}$ by $\mathbf{w}^*$ in the above expression and calling $\Delta_t = \mathbf{w}^* - \mathbf{w}_t$, we get

$$\begin{aligned}
&- \nabla f(\mathbf{w}_t)^T \Delta_{t+1} + \frac{1}{2}\Big[\Delta_{t+1}^T \hat{\mathbf{H}}_t (2\Delta_t - \Delta_{t+1})\Big] \geq 0, \\
&\Rightarrow \Delta_{t+1}^T \hat{\mathbf{H}}_t \Delta_t - \nabla f(\mathbf{w}_t)^T \Delta_{t+1} \geq \frac{1}{2}\Delta_{t+1}^T \hat{\mathbf{H}}_t \Delta_{t+1}.
\end{aligned}$$

Now, due to the optimality of $\mathbf{w}^*$, we have $\nabla f(\mathbf{w}^*)^T \Delta_{t+1} \geq 0$. Hence, we can write

$$\Delta_{t+1}^T \hat{\mathbf{H}}_t \Delta_t - (\nabla f(\mathbf{w}_t) - \nabla f(\mathbf{w}^*))^T \Delta_{t+1} \geq \frac{1}{2}\Delta_{t+1}^T \hat{\mathbf{H}}_t \Delta_{t+1}.$$

Next, substituting $\nabla f(\mathbf{w}_t) - \nabla f(\mathbf{w}^*) = \Big(\int_0^1 \nabla^2 f(\mathbf{w}^* + p(\mathbf{w}_t - \mathbf{w}^*))dp \Big)(\mathbf{w}_t - \mathbf{w}^*)$ in the above inequality, we get

$$\Delta_{t+1}^T (\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t))\Delta_t + \Delta_{t+1}^T \Big(\nabla^2 f(\mathbf{w}_t) - \int_0^1 \nabla^2 f(\mathbf{w}^* + p(\mathbf{w}_t - \mathbf{w}^*))dp\Big)\Delta_t \geq \frac{1}{2}\Delta_{t+1}^T \hat{\mathbf{H}}_t \Delta_{t+1}.$$

Using Cauchy-Schwartz inequality in the LHS above, we get

$$||\Delta_{t+1}||_2 ||\Delta_t||_2 \Big(||\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t)||_2 + \int_0^1 ||\nabla^2 f(\mathbf{w}_t) - \nabla^2 f(\mathbf{w}^* + p(\mathbf{w}_t - \mathbf{w}^*))||_2 dp \Big) \geq \frac{1}{2}\Delta_{t+1}^T \hat{\mathbf{H}}_t \Delta_{t+1}.$$

Now, using the L-Lipschitzness of $\nabla^2 f(\cdot)$ in the inequality above, we get

$$\frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1} \leq ||\Delta_{t+1}||_2||\Delta_t||_2||\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t)||_2 + \frac{L}{2}||\Delta_{t+1}||_2||\Delta_t||_2^2\int_0^1(1-p)dp,$$
$$\Rightarrow \frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1} \leq ||\Delta_{t+1}||_2\Big(||\Delta_t||_2||\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t)||_2 + \frac{L}{2}||\Delta_t||_2^2\Big). \tag{4.24}$$

Note that for the positive definite matrix $\nabla^2 f(\mathbf{w}_t) = \mathbf{A}_t^T\mathbf{A}_t$, we have $||\mathbf{A}_t||_2^2 = ||\nabla^2 f(\mathbf{w}_t)||_2$. Moreover,

$$||\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t)||_2 = ||\mathbf{A}_t^T(\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I})\mathbf{A}_t||_2 \leq ||\mathbf{A}_t||_2^2||\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I}||_2$$

Now, using Equation 4.20 from the proof of Lemma 4, we get $||\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I}||_2 = \lambda_{\max}(\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I}) \leq \epsilon$. Using this to bound the RHS of (4.24), we have, with probability at least $1 - 1/d^\tau$,

$$\frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1} \leq ||\Delta_{t+1}||_2\Big(\epsilon||\nabla^2 f(\mathbf{w}_t)||_2||\Delta_t||_2 + \frac{L}{2}||\Delta_t||_2^2\Big)$$
$$\frac{1}{2}||\mathbf{S}_t\mathbf{A}\Delta_{t+1}||_2^2 \leq ||\Delta_{t+1}||_2\Big(\epsilon||\nabla^2 f(\mathbf{w}_t)||_2||\Delta_t||_2 + \frac{L}{2}||\Delta_t||_2^2\Big),$$

where the last inequality follows from $\hat{\mathbf{H}}_t = \mathbf{A}_t^T\mathbf{S}_t^T\mathbf{S}_t\mathbf{A}_t$. Now, since the sketch dimension $m = \Omega(d^{1+\mu}/\epsilon^2)$, using Eq. (4.20) from the proof of Lemma 1 in above inequality, we get, with probability at least $1 - 1/d^\tau$,

$$\frac{1}{2}(1-\epsilon)||\mathbf{A}\Delta_{t+1}||_2^2 \leq ||\Delta_{t+1}||_2\Big(\epsilon||\nabla^2 f(\mathbf{w}_t)||_2||\Delta_t||_2 + \frac{L}{2}||\Delta_t||_2^2\Big),$$
$$\Rightarrow \frac{1}{2}(1-\epsilon)\Delta_{t+1}^T\nabla^2 f(\mathbf{w}_t)\Delta_{t+1} \leq ||\Delta_{t+1}||_2\Big(\epsilon||\nabla^2 f(\mathbf{w}_t)||_2||\Delta_t||_2 + \frac{L}{2}||\Delta_t||_2^2\Big).$$

Now, since γ and β are the minimum and maximum eigenvalues of $\nabla^2 f(\mathbf{w}^*)$, we get

$$\frac{1}{2}(1-\epsilon)||\Delta_{t+1}||_2(\gamma - L||\Delta_t||_2) \leq \epsilon(\beta + L||\Delta_t||_2)||\Delta_t||_2 + \frac{L}{2}||\Delta_t||_2^2$$

by the Lipschitzness of $\nabla^2 f(\mathbf{w})$, that is, $|\Delta_{t+1}^T(\nabla^2 f(\mathbf{w}_t) - \nabla^2 f(\mathbf{w}^*))\Delta_{t+1}| \leq L||\Delta_t||_2||\Delta||_{t+1}^2$. Rearranging for $\epsilon \leq \gamma/(8\beta) < 1/2$, we get

$$||\Delta_{t+1}||_2 \leq \frac{4\epsilon\beta}{\gamma - L||\Delta_t||_2}||\Delta_t||_2 + \frac{5L}{2(\gamma - L||\Delta_t||_2)}||\Delta_t||_2^2, \tag{4.25}$$

with probability at least $1 - 1/d^\tau$.

Let ξ_T be the event that the above inequality (in (4.25)) is true for $t = 0, 1, \cdots, T$. Thus,

$$\mathbb{P}(\xi_T) \geq \Big(1 - \frac{1}{d^\tau}\Big)^T \geq 1 - \frac{T}{d^\tau},$$

where the second inequality follows from Bernoulli's inequality. Next, assuming that the event ξ_T holds, we prove that $||\Delta_t||_2 \leq \gamma/5L$ using induction. We can verify the base case using the initialization condition, i.e. $||\Delta_0||_2 \leq \gamma/8L$. Now, assuming that $||\Delta_{t-1}||_2 \leq \gamma/5L$ and using it in the inequality (4.25), we get

$$\begin{aligned}||\Delta_t||_2 &\leq \frac{4\epsilon\beta}{\gamma} \times \frac{\gamma}{5L} + \frac{5L}{2\gamma} \times \frac{\gamma^2}{25L^2} \\ &= \frac{4\epsilon\beta}{5L} + \frac{\gamma}{10L} \\ &\leq \frac{\gamma}{L}\left(\frac{1}{10} + \frac{1}{10}\right) \leq \frac{\gamma}{5L},\end{aligned}$$

where the last inequality uses the fact that $\epsilon \leq \gamma/(8\beta)$. Thus, by induction,

$$||\Delta_t||_2 \leq \gamma/(5L) \ \ \forall \ \ t \geq 0 \quad \text{ with probability at least } 1 - T/d^\tau.$$

Using this in (4.25), we get the desired result, that is,

$$||\Delta_{t+1}||_2 \leq \frac{5\epsilon\beta}{\gamma}||\Delta_t||_2 + \frac{25L}{8\gamma}||\Delta_t||_2^2,$$

with probability at least $1 - T/d^\tau$.

4.6.3 Proof of Theorem 7

Let us define a few short notations for convenience. Say $\mathbf{g}_t = \nabla f(\mathbf{w}_t)$ and $\mathbf{H}_t = \nabla^2 f(\mathbf{w}_t) = \mathbf{A}_t^T\mathbf{A}_t$, and we know that $\hat{\mathbf{H}}_t = \mathbf{A}_t^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{A}_t$. Moreover, all the results with approximate Hessian $\hat{\mathbf{H}}_t$ hold with probability $1 - 1/d^\tau$. We skip its mention in most of the proof for brevity. The following lemmas will assist us in the proof.

Lemma 5. *M-smoothness of $f(\cdot)$ and L-Lipchitzness of $\nabla^2 f(\cdot)$ imply*

$$\left\|\nabla^2 f(\mathbf{y})\nabla f(\mathbf{y}) - \nabla^2 f(\mathbf{x})\nabla f(\mathbf{x})\right\| \leq Q||\mathbf{y} - \mathbf{x}|| \tag{4.26}$$

for all $\mathbf{x} \in \mathbb{R}^d$, $Q = (L\delta + M^2)$, where $\mathbf{y} \in \mathcal{Y}$, where $\mathcal{Y} = \{\mathbf{y} \in \mathbb{R}^d | \ ||\nabla f(\mathbf{y})|| \leq \delta\}$ and $\delta > 0$ is some constant.

Proof. We have

$$\begin{aligned}LHS &= \left\|\nabla^2 f(\mathbf{y})\nabla f(\mathbf{y}) - \nabla^2 f(\mathbf{x})\nabla f(\mathbf{x})\right\| \\ &= \left\|\nabla^2 f(\mathbf{y}) - \nabla^2 f(\mathbf{x}))\nabla f(\mathbf{y}) + \nabla^2 f(\mathbf{x})(\nabla f(\mathbf{y}) - \nabla f(\mathbf{x}))\right\|\end{aligned}$$

By applying triangle inequality and Cauchy-Schwarz to above equation, we get

$$LHS \leq ||\nabla^2 f(\mathbf{y}) - \nabla^2 f(\mathbf{x})||_2||\nabla f(\mathbf{y})|| + ||\nabla^2 f(\mathbf{x})||_2||\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})||$$

From the smoothness of $f(\cdot)$, that is, Lipshitzness of gradient, we get $||\nabla^2 f(\mathbf{x})||_2 \leq M \; \forall \; x \in \mathbb{R}^d$. Additionally, using Lipshitzness of Hessian, we get

$$\begin{aligned} LHS &\leq (L||\nabla f(\mathbf{y})|| + M^2)||\mathbf{y} - \mathbf{x}|| \\ &\leq (L\delta + M^2)||\mathbf{y} - \mathbf{x}|| \end{aligned}$$

for $\mathbf{y} \in \mathcal{Y}$. This proves the desired result. □

Lemma 6. *Let $\mathbf{A}^T = \mathbf{U}\sqrt{\Sigma}\mathbf{V}^T$ and $\mathbf{A}^T\mathbf{S}_t = \hat{\mathbf{U}}\sqrt{\hat{\Sigma}}\hat{\mathbf{V}}^T$ be the truncated Singular Value Decompositions (SVD) of $\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{S}_t$, respectively. Thus, $\mathbf{H}_t = \mathbf{U}\Sigma\mathbf{U}^T$ and $\hat{\mathbf{H}}_t = \hat{\mathbf{U}}\hat{\Sigma}\hat{\mathbf{U}}^T$. Then, for all $\mathbf{g} \in \mathbb{R}^d$, we have*

$$||\hat{\mathbf{U}}^T\mathbf{g}||^2 \geq \frac{(1-\epsilon)\eta}{M(1+\epsilon)}||\mathbf{U}^T\mathbf{g}||^2, \tag{4.27}$$

where η is defined in Assumption (5).

Proof. For all $\mathbf{g} \in \mathbb{R}^d$, using the fact that $\mathbf{A} = \mathbf{V}\sqrt{\mathbf{\Sigma}}\mathbf{U}^\mathbf{T}$, we get

$$\begin{aligned} ||\mathbf{A}\mathbf{g}||^2 &= (\mathbf{U}^T\mathbf{g})^T\Sigma(\mathbf{U}^T\mathbf{g}) \\ &\geq \lambda_{\min}(\Sigma)||\mathbf{U}^T\mathbf{g}||^2 \\ &\geq \eta||\mathbf{U}^T\mathbf{g}||^2, \end{aligned} \tag{4.28}$$

where the last inequality uses Assumption (5). In a similar fashion, we can obtain

$$\begin{aligned} ||\mathbf{S}_t^T\mathbf{A}\mathbf{g}||^2 &= (\hat{\mathbf{U}}^T\mathbf{g})^T\hat{\Sigma}(\hat{\mathbf{U}}^T\mathbf{g}) \\ &\leq \lambda_{\max}(\hat{\Sigma})||\hat{\mathbf{U}}^T\mathbf{g}||^2 \\ &\leq M(1+\epsilon)||\hat{\mathbf{U}}^T\mathbf{g}||^2, \end{aligned} \tag{4.29}$$

where the last inequality uses M-smoothness of $f(\cdot)$ and Lemma 4. Also, from the subspace embedding property of $\mathbf{S}_t$ (see Lemma 4), we have

$$||\mathbf{S}^T\mathbf{A}\mathbf{g}||^2 \geq (1-\epsilon)||\mathbf{A}\mathbf{g}||^2.$$

Now, using the above inequality and Eqs. (4.28) and (4.29), we get

$$||\hat{\mathbf{U}}^T\mathbf{g}||^2 \geq \frac{(1-\epsilon)\eta}{M(1+\epsilon)}||\mathbf{U}^T\mathbf{g}||^2, \tag{4.30}$$

which is the desired result. □

Now we are ready to prove Theorem 7. Let $\mathbf{H}_t = \mathbf{U}\Sigma\mathbf{U}^T$ and $\hat{\mathbf{H}}_t = \hat{\mathbf{U}}\hat{\Sigma}\hat{\mathbf{U}}^T$ be the truncated SVDs of $\mathbf{H}_t$ and $\hat{\mathbf{H}}_t$, respectively. Also, let α_t be the step-size obtained using line-search in (4.6) in the t-th iteration. Thus, Eq. (4.6) with the update direction $\mathbf{p}_t = -\hat{\mathbf{H}}_t^\dagger\mathbf{g}_t$ implies

$$\begin{aligned} ||\mathbf{g}_{t+1}||^2 &\leq ||\mathbf{g}_t||^2 - 2\beta\alpha_t\langle\hat{\mathbf{H}}_t\mathbf{g}_t, \hat{\mathbf{H}}_t^\dagger\mathbf{g}_t\rangle \\ &= ||\mathbf{g}_t||^2 - 2\beta\alpha_t||\hat{\mathbf{U}}_T^T\mathbf{g}_t||^2, \end{aligned} \tag{4.31}$$

where the last equality uses the fact that $\hat{\mathbf{H}}_t^\dagger$ can be expressed as $\hat{\mathbf{H}}_t^\dagger = \hat{\mathbf{U}}\hat{\Sigma}^{-1}\hat{\mathbf{U}}^T$. Note that Lemma 5 implies that the function $||\nabla f(\mathbf{y})||^2/2$ is smooth for all $y \in \mathcal{Y}$, where $\mathcal{Y} = \{\mathbf{y} \in \mathbb{R}^d | \; ||\nabla f(\mathbf{y})|| \leq \delta\}$. Smoothness in turn implies the following property (see [157], Theorem 2.1.10)

$$\frac{1}{2}||\nabla f(\mathbf{y})||^2 \leq \frac{1}{2}||\nabla f(\mathbf{x})||^2 + \langle \nabla^2 f(\mathbf{x}), \mathbf{y} - \mathbf{x}\rangle + \frac{1}{2}Q||\mathbf{y} - \mathbf{x}||^2 \; \forall \, \mathbf{x}, \mathbf{y} \in (Y), \tag{4.32}$$

where $Q = L\delta + M^2$. We take $\delta = ||\nabla f(\mathbf{w}_0)||$ where $\mathbf{w}_0$ is the initial point of our algorithm. Due to line-search condition in (4.6), it holds that $||\nabla f(\mathbf{w}_t)|| \leq ||\nabla f(\mathbf{w}_0)|| \; \forall \; t > 0$. Thus, substituting $\mathbf{x} = \mathbf{w}_t$ and $\mathbf{y} = \mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \mathbf{p}_t$, we get

$$\begin{aligned} \frac{1}{2}||\mathbf{g}_{t+1}||^2 &\leq \frac{1}{2}||\mathbf{g}_t||^2 + \langle \mathbf{H}_t\mathbf{g}_t, \alpha_t\mathbf{p}_t\rangle + \frac{1}{2}Q\alpha^2||\mathbf{p}_t||^2 \\ \Rightarrow ||\mathbf{g}_{t+1}||^2 &\leq ||\mathbf{g}_t||^2 + \langle 2\mathbf{H}_t\mathbf{g}_t, \alpha_t\mathbf{p}_t\rangle + Q\alpha^2||\mathbf{p}_t||^2, \end{aligned} \tag{4.33}$$

where

$$Q = L||\nabla f(\mathbf{w}_0)|| + M^2.$$

Also, since the minimum non-zero eigenvalue of $\mathbf{H}_t \geq \eta$ from Assumption (5), the minimum non-zero eigenvalue of $\hat{\mathbf{H}}_t$ is at least $\eta - \epsilon M$ from Lemma 4. Thus,

$$|\hat{\mathbf{H}}_t^\dagger||_2 \leq 1/(\eta - \epsilon M). \tag{4.34}$$

|Moreover,

$$||\mathbf{p}_t|| = || - \hat{\mathbf{H}}_t^\dagger\mathbf{g}_t|| \leq ||\hat{\mathbf{H}}_t^\dagger||_2||\mathbf{g}_t|| \leq \frac{||\mathbf{g}_t||}{(\eta - \epsilon M)}. \tag{4.35}$$

Using this in (4.33), we get

$$||\mathbf{g}_{t+1}||^2 \leq ||\mathbf{g}_t||^2 - 2\alpha_t\langle \mathbf{H}_t\mathbf{g}_t, \hat{\mathbf{H}}_t^\dagger\mathbf{g}_t\rangle + Q\alpha^2\frac{||\mathbf{g}_t||^2}{(\eta - \epsilon M)^2}. \tag{4.36}$$

Now,

$$\begin{aligned} -\langle \mathbf{H}_t\mathbf{g}_t, \hat{\mathbf{H}}_t^\dagger\mathbf{g}_t\rangle &= -\langle \hat{\mathbf{H}}_t\mathbf{g}_t, \hat{\mathbf{H}}_t^\dagger\mathbf{g}_t\rangle + \langle (\hat{\mathbf{H}}_t - \mathbf{H}_t)\mathbf{g}_t, \hat{\mathbf{H}}_t^\dagger\mathbf{g}_t\rangle \\ \Rightarrow -\langle \mathbf{H}_t\mathbf{g}_t, \hat{\mathbf{H}}_t^\dagger\mathbf{g}_t\rangle &\leq -||\hat{\mathbf{U}}_t^T\mathbf{g}_t||^2 + ||\mathbf{g}_t||^2||\hat{\mathbf{H}}_t - \mathbf{H}_t||_2||\hat{\mathbf{H}}_t^\dagger||_2, \end{aligned}$$

where the last inequality is obtained by applying the triangle inequality and Cauchy-Schwartz inequality. This can be further simplified using Lemma 4 and Eq. (4.34) as

$$-\langle \mathbf{H}_t\mathbf{g}_t, \hat{\mathbf{H}}_t^\dagger\mathbf{g}_t\rangle \leq -||\hat{\mathbf{U}}_t^T\mathbf{g}_t||^2 + \frac{\epsilon M}{(\eta - M\epsilon)}||\mathbf{g}_t||^2$$

Using the above in Eq. (4.36), we get

$$||\mathbf{g}_{t+1}||^2 \leq ||\mathbf{g}_t||^2 + 2\alpha_t(-||\hat{\mathbf{U}}_t^T\mathbf{g}_t||^2 + \frac{\epsilon M}{(\eta - M\epsilon)}||\mathbf{g}_t||^2) + Q\alpha_t^2\frac{||\mathbf{g}_t||^2}{(\eta - \epsilon M)^2} \tag{4.37}$$

Note that the upper bound in Eq. (4.37) always holds. Also, we want the inequality in (4.31) to hold for some $\alpha_t > 0$. Therefore, we want α_t to satisfy the following (and hope that it is always satisfied for some $\alpha_t > 0$)

$$
\begin{aligned}
&||\mathbf{g}_t||^2 + 2\alpha_t(-||\hat{\mathbf{U}}_t^T\mathbf{g}_t||^2 + \frac{\epsilon M}{(\eta - M\epsilon)}||\mathbf{g}_t||^2) + Q\alpha_t^2\frac{||\mathbf{g}_t||^2}{(\eta-\epsilon M)^2} \leq ||\mathbf{g}_t||^2 - 2\beta\alpha_t||\hat{\mathbf{U}}_T^T\mathbf{g}_t||^2 \\
&\Rightarrow Q\alpha_t^2\frac{||\mathbf{g}_t||^2}{(\eta-\epsilon M)^2} \leq 2\alpha_t\left[(1-\beta)||\hat{\mathbf{U}}_t^T\mathbf{g}_t||^2 - \frac{\epsilon M}{(\eta - M\epsilon)}||\mathbf{g}_t||^2\right] \\
&\Rightarrow \alpha_t \leq \frac{2(\eta-\epsilon M)^2}{Q}\left[(1-\beta)\frac{||\hat{\mathbf{U}}_t^T\mathbf{g}_t||^2}{||\mathbf{g}_t||^2} - \frac{\epsilon M}{(\eta - M\epsilon)}\right]. \qquad (4.38)
\end{aligned}
$$

Thus, any α_t satisfying the above inequality would definitely satisfy the line-search termination condition in

Now, using Lemma 6 and Assumption (6), we have

$$
||\hat{\mathbf{U}}_t^T\mathbf{g}_t||^2 \geq \frac{(1-\epsilon)\eta}{M(1+\epsilon)}||\mathbf{U}_t^T\mathbf{g}||^2 \geq \frac{(1-\epsilon)\eta}{M(1+\epsilon)}\nu||\mathbf{g}||^2. \qquad (4.39)
$$

Using the above in Eq. (4.38) to find an iteration independent bound on α_t, we get

$$
\alpha_t \leq \frac{2(\eta-\epsilon M)^2}{Q}\left[(1-\beta)\nu - \frac{\epsilon M}{(\eta - M\epsilon)}\right]. \qquad (4.40)
$$

Hence, line-search will always terminate for all α_t that satisfy the above inequality. This can be further simplified by assuming that ϵ is small enough such that $\epsilon < \eta/2M$. Thus, $\eta - M\epsilon > \eta/2$, and the sufficient condition on α_t in (4.40) becomes

$$
\alpha_t \leq \frac{\eta}{2Q}\big[(1-\beta)\nu\eta - 2\epsilon M\big]. \qquad (4.41)
$$

For a positive α_t to always exist, we require ϵ to further satisfy

$$
\epsilon \leq \frac{(1-\beta)\nu\eta}{2M}, \qquad (4.42)
$$

which is tighter than the initial upper bound on ϵ. Now, Eqs. (4.31) and (4.39) proves the desired result, that is

$$
||\mathbf{g}_{t+1}||^2 \leq ||\mathbf{g}_t||^2 - 2\beta\alpha_t||\hat{\mathbf{U}}_T^T\mathbf{g}_t||^2 \leq \left(1 - 2\beta\alpha_t\nu\frac{(1-\epsilon)\eta}{M(1+\epsilon)}\right)||\mathbf{g}_t||^2.
$$

Thus, OverSketched Newton for the weakly-convex case enjoys a uniform linear convergence rate of decrease in $||\nabla f(\mathbf{w})||^2$.

4.7 Conclusions

We proposed OverSketched Newton, a straggler-resilient distributed optimization algorithm for serverless systems. It uses the idea of matrix sketching from RandNLA to find an approximate second-order update in each iteration. We proved that OverSketched Newton has a local linear-quadratic convergence rate for the strongly-convex case, where the dependence on the linear term can be made to diminish by increasing the sketch dimension. Moreover, it has a linear global convergence rate for weakly-convex functions. By exploiting the massive scalability of serverless systems, OverSketched Newton produces a global approximation of the second-order update. Empirically, this translates into faster convergence than state-of-the-art distributed optimization algorithms on AWS Lambda.

Chapter 5

LocalNewton

In this chapter, we propose LocalNewton, a distributed second-order algorithm with *local averaging* to address the communication bottleneck problem in distributed optimization.

5.1 Introduction

An explosion in data generation and data collection capabilities in recent years has resulted in the segregation of computing and storage resources. Distributed machine learning is one example where each worker machine processes only a subset of the data, while the master machine coordinates with workers to learn a good model. Such coordination can be time-consuming since it requires frequent communication between the master and worker nodes, especially for systems that have large compute resources, but are bottlenecked by communication costs.

Communication costs in distributed optimization can be broadly classified into two types—(a) latency cost and (b) bandwidth cost [5], [132]. Latency is the fixed cost associated with sending a message, and it is generally independent of the size of the message. Bandwidth cost, on the other hand, is directly proportional to the size of the message. Many recent works have focused on reducing the bandwidth cost by reducing the size of the gradient or the model to be communicated using techniques such as sparsification [93], [158], sketching [54], [100] and quantization [90], [96], [159]–[161]. Schemes that perform inexact updates in each iteration, however, can *increase* the number of iterations required to converge to the same quality model, both theoretically and empirically [158], [162]. This can, in turn, increase the total training time in systems where latency costs dominate bandwidth costs.

Serverless frameworks—such as Amazon Web Services (AWS) Lambda and Microsoft Azure Functions—are an example of a setting where high communication latency between worker machines dominates the running time of the algorithm [4], [6]. These systems use cloud storage (e.g., AWS S3) to store enormous amounts of data, while using a large number of low-quality workers for large-scale computation. Naturally, the communication between the high-latency storage and the commodity workers is extremely slow [e.g., see 1], resulting in impractical end-to-end times for many popular optimization algorithms such as SGD [4], [6]. Furthermore, communication

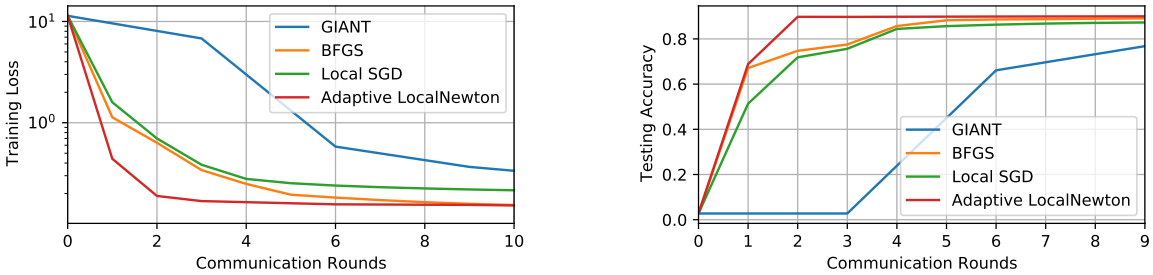


Figure 5.1: Training loss and testing accuracy versus communication rounds for Adaptive LocalNewton and existing schemes for communication-efficient optimization.

failures between the cloud storage and serverless workers consistently give rise to stragglers, and this introduces synchronization delays [147], [163].

Yet another setting where latency costs may outweigh bandwidth costs is federated learning, where the computation is performed locally at the mobile device (which is generally the source of the data) due to a high-cost barrier in transferring the data to traditional computing platforms [54], [56]. Such mobile resources (e.g., mobile phones, wearable devices, etc.) have reasonable compute power, but they can be severely limited by communication latency (e.g., inadequate network connection to synchronize frequent model updates). For this reason, schemes like Local Stochastic Gradient Descent (Local SGD) have become popular, since they try to mitigate the communication costs by performing more *local* computation at the worker machines, thus substantially reducing the number of communication rounds required [55].

These trends suggest that optimization schemes that reduce communication rounds between workers are highly desirable. In this chapter, we go one-step forward—we propose and analyze a second order method with *local* computations. Being a second order algorithm, the iteration complexity of LocalNewton is inherently low. Moreover, its local nature further cuts down the communication cost between the worker and the master node. To the best of our knowledge, this is the first work to propose and analyze a second order optimization algorithm with local averaging.

Additionally, in this chapter, we introduce an adaptive variant of the LocalNewton algorithm, namely Adaptive LocalNewton. This algorithm chooses the number of local iterations adaptively at the master after each communication round by observing the change in training loss. Thus, it further refines the iterates obtained through LocalNewton by adaptively and successively reducing the number of local iterations, thereby improving the quality of the model updates. Furthermore, when the number of local iterations, L, reduces to 1, Adaptive LocalNewton automatically switches to a standard second order optimization algorithm, namely GIANT, proposed in [51].

In Sec. 5.4, we show that the iterates of LocalNewton converge to a norm ball (with small radius) around the global minima. From this point of view, we may think of LocalNewton as an initialization algorithm, rather than an optimization one; since it takes the iterates close to the optimal point in only a few rounds of communication. After reaching sufficiently close to the solution point, one may choose some standard optimization algorithm to reach to the solution. In Adaptive LocalNewton, we first exploit the fast convergence of LocalNewton ($L \geq 1$), and afterwards, when $L = 1$, Adaptive LocalNewton switches to the standard optimization algorithm

(e.g., GIANT [51]).

Our contributions. Inspired by recent progress in local optimization methods (that reduce communication cost by limiting the frequency of synchronization) and distributed and stochastic second order methods (that use the local curvature information), we propose a local second-order algorithm called LocalNewton. The proposed LocalNewton method saves on communication costs in two ways. First, it updates the models at the master only sporadically, thus requiring only one communication round per multiple iterations. Second, it uses the second-order information to reduce the number of iterations, and hence it reduces the overall rounds of communication.

Important features of LocalNewton include:

1. *Simplicity*: In LocalNewton, each worker takes only a few Newton steps [149] on local data, agnostic of other workers. These local models are then averaged once every $L(\geq 1)$ iterations at the master node.

2. *Practicality*: Unlike many first-order and distributed second-order schemes, LocalNewton does not require hyperparameter tuning for step-size, mini-batch size, etc., and the only hyperparameter required is the number of local iterations L. We also propose Adaptive LocalNewton, an adaptive version which automatically reduces L as the training proceeds by monitoring the training loss at the master.

3. *Convergence guarantees*: In general, proving convergence guarantees for local algorithms is not straightforward. Only recently, it has been proved [57] that local SGD converges as fast as SGD, thereby explaining the well-studied empirical successes [54]. In this chapter, we develop novel techniques to highlight the convergence behaviour of LocalNewton.

4. *Reduced training times*: We implement LocalNewton on the serverless environment AWS Lambda using the Pywren framework [1]. Through extensive empirical evaluation, we show that the significant savings in terms of communication rounds translate to savings in running time on this high-latency distributed computing environment.

5. *Adaptivity*: We propose Adaptive LocalNewton, which is an adaptive variant of the LocalNewton algorithm. In the adaptive scheme, based on the change in function objective value, the master modulates the number of local iterations at the worker machines. This improves the quality of the model updates as discussed in detail in Sec. 5.3.2.

6. *LocalNewton as an initialization algorithm*: Since the convergence guarantees of LocalNewton (see Sec. 5.4) only ensure that the iterates stay in a norm ball around the minima, one may rethink LocalNewton as an initialization algorithm, rather than an optimization one. In only a very few communication rounds, LocalNewton takes the iterates very close to the optimal solution. After that, our algorithm switches to a standard second-order algorithm, GIANT [51].

Fig. 5.1 illustrates savings due to Adaptive LocalNewton, where we plot training loss and test accuracy with communication rounds, for several popular communication-efficient schemes for logistic regression on the w8a dataset [119] (see Sec. 5.5 for a details on experiments). Observe that Adaptive LocalNewton reaches close to the optimal training loss very quickly, when compared to schemes like Local SGD [54], [57], GIANT [51] and BFGS [164].

5.2 Problem Formulation

We first define the notation used and then introduce the basic problem setup considered in this chapter.

Notation. Throughout the chapter, vectors (e.g., $\mathbf{g}$) and matrices (e.g., $\mathbf{H}$) are represented as bold lowercase and uppercase letters, respectively. For a vector $\mathbf{g}$, $\|\mathbf{g}\|$ denotes its ℓ_2 norm, and $\|\mathbf{H}\|_2$ denotes the spectral norm of matrix $\mathbf{H}$. The identity matrix is denoted as $\mathbf{I}$, and the set $\{1, 2, \cdots, n\}$ is denoted as $[n]$, for all positive integers n. Further, we use superscript (e.g., $\mathbf{g}^k$) to denote the worker index and subscript (e.g., $\mathbf{g}_t$) to denote the iteration counter (i.e., time index), unless stated otherwise.

Problem Setup. We are interested in solving empirical risk minimization problems of the following form in a distributed fashion

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ f(\mathbf{w}) \triangleq \frac{1}{n}\sum_{j=1}^{n} f_j\left(\mathbf{w}\right) \right\}, \tag{5.1}$$

where $f_j(\cdot) : \mathbb{R}^d \to \mathbb{R}$, for all $j \in [n] = \{1, 2, \cdots, n\}$, models the loss of the j-th observation given an underlying parameter estimate $\mathbf{w} \in \mathbb{R}^d$. In machine learning, such problems arise frequently, e.g., logistic and linear regression, support vector machines, neural networks and graphical models. Specifically, in the case of logistic regression,

$$f_j(\mathbf{w}) = \ell_j(\mathbf{w}^T\mathbf{x}_j) = \log(1 + e^{-y_j\mathbf{w}^T\mathbf{x}_j}) + \frac{\gamma}{2}\|\mathbf{w}\|^2,$$

where $\ell_j(\cdot)$ is the loss function for sample $j \in [n]$ and γ is an appropriately chosen regularization parameter. Also, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ is the sample matrix containing the input feature vectors $\mathbf{x}_j \in \mathbb{R}^d, j \in [n]$, and $\mathbf{y} = [y_1, y_2, \cdots, y_n]$ is the corresponding label vector. Hence, $(\mathbf{x}_j, y_j)$ together define the j-th observation and $(\mathbf{X}, \mathbf{y})$ define the training dataset.

For such problems, the gradient and the Hessian at the t-th iteration are given by

$$\mathbf{g}_t = \nabla f\left(\mathbf{w}_t\right) = \frac{1}{n}\sum_{j=1}^{n} \nabla f_j(\mathbf{w}_t) \text{ and}$$

$$\mathbf{H}_t = \nabla^2 f\left(\mathbf{w}_t\right) = \frac{1}{n}\sum_{j=1}^{n} \nabla^2 f_j\left(\mathbf{w}_t\right),$$

respectively, where $\mathbf{w}_t$ is the model estimate at the t-th iteration.

Data distribution at each worker: Let there be a total of K workers. We assume that the k-th worker is assigned a subset $\mathcal{S}_k \subset [n]$, for all $k \in [K] = \{1, 2, \cdots, K\}$, of the n data points, chosen uniformly at random without replacement.[1] Let the number of samples at each worker be

[1]This corresponds simply to partitioning the dataset and assigning an equal number of observations to each worker, if the observations are independent and identically distributed. If not, randomly shuffling the observations and then performing a data-independent partitioning is equivalent to uniform sampling without replacement.

$s = |\mathcal{S}_k| \; \forall \; k \in [K]$, where $s \ll n$ in practice. Also, by the virtue of sampling without replacement, we have $\mathcal{S}_1 \cup \mathcal{S}_2 \cup \cdots \cup \mathcal{S}_K = [n]$ and $\mathcal{S}_i \cap \mathcal{S}_j = \Phi$ for all $i, j \in [K]$. Hence, the number of workers is given by $K = n/s$.

Next, we describe LocalNewton, a communication-efficient algorithm for distributed optimization.

5.3 Algorithms

In this section, we propose two novel algorithms for distributed optimization. First, we propose LocalNewton, a second order algorithm with local averaging. Subsequently, we also propose an adaptive variant of LocalNewton. Here LocalNewton acts as a good initialization scheme that pushes its iterates close to the optimal solution in a small number of communication rounds. Finally, a standard second-order algorithm is used to converge to the optimal solution.

5.3.1 LocalNewton

We consider synchronous second-order methods for distributed learning, where local models are synced after every L iterations. Let $\mathcal{I}_t \subseteq [t]$ be the set of indices where the model is synced, that is, $\mathcal{I}_t = [0, L, 2L, \cdots, t_0]$, where t_0 is the last iteration just before t where the models were synced.

At the k-th worker in the t-th iteration, the local function value (at the local iterate $\mathbf{w}_t^k$) is

$$f^k(\mathbf{w}_t^k) = \frac{1}{s} \sum_{j \in \mathcal{S}_k} f_j(\mathbf{w}_t^k). \tag{5.2}$$

The k-th worker tries to minimize the local function value in Eq (5.2) in each iteration. The corresponding local gradient $\mathbf{g}_t^k$ and local Hessian $\mathbf{H}_t^k$, respectively, at k-th worker in t-th iteration can be written as

$$\mathbf{g}_t^k = \nabla f^k(\mathbf{w}_t^k) = \frac{1}{s} \sum_{j \in \mathcal{S}_k} \nabla f_j(\mathbf{w}_t^k) \text{ and}$$
$$\mathbf{H}_t^k = \nabla^2 f^k(\mathbf{w}_t^k) = \frac{1}{s} \sum_{j \in \mathcal{S}_k} \nabla^2 f_j(\mathbf{w}_t^k).$$

Let us consider the following LocalNewton update at the k-th worker and $(t+1)$-st iteration:

$$\mathbf{w}_{t+1}^k = \begin{cases} \mathbf{w}_t^k - \alpha_t^k \mathbf{H}^k(\mathbf{w}_t^k)^{-1} \mathbf{g}^k(\mathbf{w}_t^k), & \text{if } t \notin \mathcal{I}_t \\ \bar{\mathbf{w}}_t - \alpha_t^k \mathbf{H}^k(\bar{\mathbf{w}}_t)^{-1} \mathbf{g}^k(\bar{\mathbf{w}}_t), & \text{if } t \in \mathcal{I}_t, \end{cases} \tag{5.3}$$

where $\bar{\mathbf{w}}_t = \frac{1}{K} \sum_{k=1}^K \mathbf{w}_t^k \; \forall \; t$, and α_t^k is the step-size at the k-th worker at iteration t.[2]

[2]In practice, one need not calculate the exact $\mathbf{H}^k(\mathbf{w}_t^k)^{-1}\mathbf{g}^k(\mathbf{w}_t^k)$, and efficient algorithms like conjugate gradient descent can be used [152].

Algorithm 12: LocalNewton

Input : Local function $f^k(\cdot)$ at the k-th worker; Initial iterate $\bar{\mathbf{w}}_0 \in \mathbb{R}^d$; Line search parameter $0 < \beta \leq 1/2$; Number of iterations T, Set $\mathcal{I}_T \subseteq \{1, 2, \cdots, T\}$ where models are synced

for $k = 1$ *to* K *in parallel* **do**
  Initialization: $\mathbf{w}_0^k = \bar{\mathbf{w}}_0$
  for $t = 0$ *to* $T - 1$ **do**
    if $t \in \mathcal{I}_T$ **then**
      // Master averages the local models
      $\bar{\mathbf{w}}_t = \frac{1}{K} \sum_{k=1}^{K} \mathbf{w}_t^k$
      $\mathbf{w}_t^k = \bar{\mathbf{w}}_t$
    end
    // Compute the local gradient
    $\mathbf{g}^k(\mathbf{w}_t^k) = \nabla f^k(\mathbf{w}_t^k)$.
    // Compute the update direction
    $\mathbf{p}_t^k = \mathbf{H}^k(\mathbf{w}_t^k)^{-1}\mathbf{g}^k(\mathbf{w}_t^k)$
    Find step-size α_t^k using line search (Eq. (5.5))
    Update model: $\mathbf{w}_{t+1}^k = \mathbf{w}_t^k - \alpha_t^k \mathbf{p}_t^k$
  end
end

Also, define the local descent direction at the k-th worker at iteration t as

$$\mathbf{p}_t^k = \alpha_t^k \mathbf{H}^k(\mathbf{w}_t^k)^{-1}\mathbf{g}^k(\mathbf{w}_t^k)$$

and similarly define

$$\bar{\mathbf{p}}_t = \frac{1}{K} \sum_{k=1}^{K} \mathbf{p}_t^k.$$

We can see that $\bar{\mathbf{w}}_{t+1} = \bar{\mathbf{w}}_t - \bar{\mathbf{p}}_t$. Detailed steps for LocalNewton are provided in Algorithm 12.

Note that $\bar{\mathbf{w}}_t$ is not explicitly calculated for all t, but only for $t \in \mathcal{I}_t$. However, we will use the technique of perturbed iterate analysis and show the convergence of the sequence $f(\bar{\mathbf{w}}_1), \cdots, f(\bar{\mathbf{w}}_t)$ to $f(\mathbf{w}^*)$. In the next section, we present Adaptive LocalNewton, an algorithm to adaptively choose the number of local iterations, L.

5.3.2 Adaptive LocalNewton

Motivating Example (Least-squares). Let us now consider the simple example of unregularized linear least squares, i.e., the loss function at the k-th worker is $f^k(\mathbf{w}) = \frac{1}{s}\|\mathbf{y}^k - \mathbf{X}^k\mathbf{w}\|^2$, where $\mathbf{y}^k = [y_1^k, y_2^k, \ldots, y_s^k]^\top \in \mathbb{R}^s$ and $\mathbf{X} = [\mathbf{x}_1^k, \ldots, \mathbf{x}_s^k]^\top \in \mathbb{R}^{s \times d}$. Note that one second-order

Algorithm 13: Adaptive LocalNewton

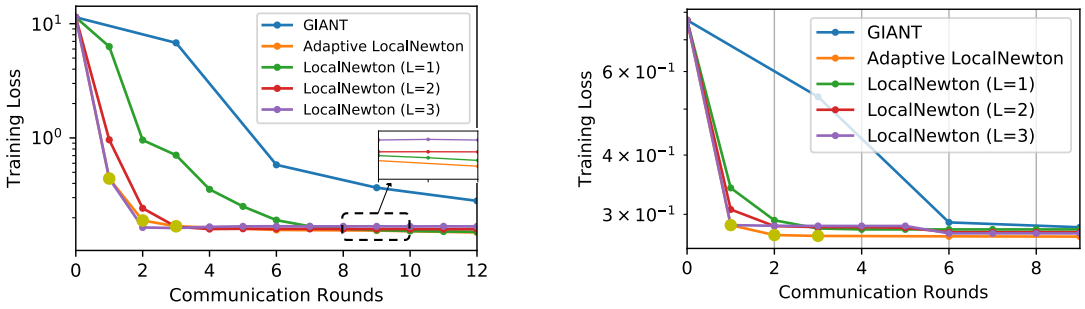
Input : Minimum decrement $\delta(>0)$ in global loss function;

Initialization: $f_{prev} = f(\bar{\mathbf{w}}_0)$, Number of local iterations L, Set $\mathcal{I}_T \subseteq \{1, 2, \cdots, T\}$ where models are synced every L local iterations
for $t = 1$ *to* $T-1$ **do**
  // k-th worker runs LocalNewton locally to get $\mathbf{w}_t^k$
  Workers run Algorithm 12
  if $t \in \mathcal{I}_T$ **then**
    // Master averages the local models
    $\bar{\mathbf{w}}_t = \frac{1}{K}\sum_{k=1}^{K} \mathbf{w}_t^k$
    if $f_{prev} - f(\bar{\mathbf{w}}_t) < \delta$ *and* $L \geq 1$ **then**
      // The global function did not decrease enough
      if $L = 1$ **then**
        Switch to GIANT [51]
      end
      else
        Decrease L: $L = L - 1$
        Update $\mathcal{I}_T$ according to the new value of L
      end
    end
    $f_{prev} = f(\bar{\mathbf{w}}_t)$
  end
end

iteration (with step-size one) reaches the optimal solution, say $(\mathbf{w}^k)^* = \mathbf{w}_t^k - (\mathbf{H}_t^k)^{-1}\mathbf{g}^k\mathbf{w}_t^k = [(\mathbf{X}^k)^T\mathbf{X}^k]^{-1}(\mathbf{X}^k)^T\mathbf{y}^k \; \forall \; \mathbf{w}_t^k \in \mathbb{R}^d$, for the local loss function $f^k(\mathbf{w})$ at the k-th worker.

Thus, applying LocalNewton here with $L \geq 1$ would imply that the local iterates at the k-th worker are fixed at $\mathbf{w}_t^k = (\mathbf{w}^k)^*$ while the global iterates at the master are fixed at $\bar{\mathbf{w}}_t = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{w}^k)^*$ for all $t \in [T]$. Note that $\bar{\mathbf{w}}_t \neq \bar{\mathbf{w}}^*$ in general, where $\bar{\mathbf{w}}^*$ is the optimal solution for the global problem in Eq. (5.1). Hence, LocalNewton (Algorithm 12) does not reach the optimal solution for unregularized least-squares. In fact, in Theorems 8 and 9, we show that running LocalNewton algorithm results in an error floor of the order $1/\sqrt{s}$ for any convex loss function.

Motivated from the above example, if we want to attain the optimal solution, one needs to switch to optimization algorithms which yield no error floor. One standard example of such an algorithm is GIANT ([51]), which is a communication-efficient distributed second-order algorithm. However, for general convex functions, the convergence of GIANT requires the initial point to be close to the optimal solution. From this point of view, one can think of LocalNewton as an initialization scheme that pushes the iterates close to the solution (within a radius of $\mathcal{O}(1/\sqrt{s})$) with a few rounds of communication. Then, one can switch to the GIANT algorithm to obtain convergence to the solution point. We call this algorithm Adaptive LocalNewton, where the master modulates the value



(a) w8a dataset (b) EPSILON dataset

Figure 5.2: Comparing LocalNewton (for different values of L) and GIANT. In general, LocalNewton reaches very close to the optimal solution. In that region, however, the convergence rate of LocalNewton is slow. This is mitigated by using Adaptive LocalNewton which appends the LocalNewton iterations with better quality (but more expensive) updates from GIANT.

of L successively over iterations and finally switches to GIANT to obtain the final solution. The details are given in Algorithm 13.

Recall that GIANT synchronizes the local gradients and the local descent direction in every iteration [51]. Further, it finds the step-size by doing a distributed backtracking line-search requiring an additional round of communication (Sec. 5.2, [51]). Finally, the master updates the model by using the average descent direction and the obtained step-size and ships the model to all the workers. Thus, each iteration in GIANT requires three rounds of communication. This approach has compared favorably to other popular distributed second-order methods (e.g., DANE [48], AGD [157], BFGS [164], CoCoA [53], DiSCO [49]).

In Fig. 5.2, we compare GIANT to LocalNewton, where LocalNewton is run with 100 workers for $L = 1, 2$ and 3, for two datasets—w8a and EPSILON obtained from LIBSVM [119]. Note that LocalNewton converges much faster with respect to communication rounds for all the three datasets since it communicates intermittently, i.e., once every few local second-order iterations (e.g., after 3 local iterations for $L = 3$). Not shown here is that testing accuracy follows the same trends. Further, the quality of the final solution improves as we reduce L. However, it reaches extremely close to the optimal training loss, but it converges very slowly (or flattens out) after that.

These empirical observations further motivate Adaptive LocalNewton: a second-order distributed algorithm that adapts the number of local iterations as the training progresses and ultimately finishes with GIANT. This can be done by monitoring the objective function at the master, e.g., reduce L if the loss stops improving (or switch to GIANT if $L = 1$).[3] See Algorithm 13, where we provide the pseudo-code for Adaptive LocalNewton. Whenever the global function value at the master does not decrease more than a constant δ, we decrease the value of L to improve the quality of second order estimate. In this sense, Adaptive LocalNewton can be seen as providing a carefully-constructed or

[3]To further reduce the communication rounds and dependency on L, each worker can update the model for multiple values of L and send the concatenated model updated to the master. The master can decide the right value of L by evaluating the loss/accuracy for these different models.

gradually-annealed initialization for GIANT.

Comparison with GIANT: Algorithm 13 is further motivated by the theoretical guarantees we obtain in the subsequent section. In Theorems 8 and 9 of Sec. 5.4, we prove the convergence of LocalNewton to the optimal solution within an error floor starting with any initial point in $\mathbb{R}^d$. In sharp contrast, Theorem 2 in GIANT [51], the authors convergence guarantees to the optimal solution when the current model is sufficiently close to the optimal model. From Fig. 5.2, we see that Adaptive LocalNewton significantly outperforms GIANT in terms of rounds of communication. Adaptive LocalNewton starts from L=3, and yellow dots in its plot denote the reduction in the value of L by one or a switch to GIANT if $L = 1$.

5.4 Convergence Guarantees

In this section, we present the main theoretical contributions of the chapter. For this, we only consider the (non-adaptive) LocalNewton algorithm. Obtaining theoretical guarantees for Adaptive LocalNewton is kept as an interesting future work.

First, we delineate some assumptions on $f(\cdot)$ required to prove theoretical convergence of the proposed method.

Assumptions: We make the following standard assumptions on the objective function $f(\cdot)$ for all $\mathbf{w} \in \mathbb{R}^d$:

1. $f_i(\cdot)$, for all $i \in [n]$, is twice differentiable.
2. $f(\cdot)$ is κ-strongly convex, that is, $\nabla^2 f(\mathbf{w}) \succcurlyeq \kappa \mathbf{I}$.
3. $f(\cdot)$ is M-smooth, that is, $\nabla^2 f(\mathbf{w}) \preccurlyeq M\mathbf{I}$.
4. $\|\nabla^2 f_i(\cdot)\|_2, i \in [n]$, is upper bounded. That is, $\nabla^2 f_i(\mathbf{w}) \preccurlyeq B\mathbf{I}$, for all $i \in [n]$.

In the following lemma, we make use of matrix concentration inequalities to show that, for sufficiently large sample size s, the local Hessian at each worker is also strongly convex and smooth with high probability.

Lemma 7. *Let $f(\cdot)$ satisfy assumptions 1-4 and $0 < \epsilon \leq 1/2$ and $0 < \delta < 1$ be fixed constants. Then, if $s \geq \frac{4B}{\kappa\epsilon^2} \log \frac{2d}{\delta}$, the local Hessian at the k-th worker satisfies*

$$(1-\epsilon)\kappa \preccurlyeq \nabla^2 f^k(\mathbf{w}) = \mathbf{H}^k(\mathbf{w}) \preccurlyeq (1+\epsilon)M, \tag{5.4}$$

for all $\mathbf{w} \in \mathbb{R}^d$ and $k \in [K]$ with probability at least $1 - \delta$.

Proof. See Appendix 5.6.1. □

Step-size selection: Let each worker locally choose a step-size according to the following rule

$$\alpha_t^k = \max_{\alpha \leq \alpha^\star} \alpha \quad \text{such that}$$
$$f^k\left(\mathbf{w}_t^k - \alpha \mathbf{p}_t^k\right) \leq f^k\left(\mathbf{w}_t^k\right) - \alpha\beta(\mathbf{p}_t^k)^T \nabla f^k\left(\mathbf{w}_t^k\right), \tag{5.5}$$

for some constant $\beta \in (0, 1/2]$, where the parameter $\alpha^\star (\leq 1)$ depends on the properties of the objective function:[4]

$$\alpha^\star \leq \min \left\{ \frac{(1-\beta)\kappa}{M}, \frac{2\beta\kappa^2}{3M[M-\kappa/4]} \right\}. \tag{5.6}$$

Now, we are almost ready to prove the main theorems—Theorem 8 discusses the case when $L = 1$, and Theorem 9 discusses the case when $L > 1$. Before that, let us state the following auxiliary lemma which is required to prove the main theorems.

Lemma 8. *Let the function $f(\cdot)$ satisfy assumptions 1-3, and suppose that step-size α_t^k satisfies the line-search condition in* (5.5)*. Also, let $0 < \epsilon < 1/2$ and $0 < \delta < 1$ be fixed constants. Moreover, let the sample size $s \geq \frac{4B}{\kappa\epsilon^2} \log \frac{2d}{\delta}$. Then, the LocalNewton update, defined in Eq.* (5.3)*, at the k-th worker satisfies*

$$f^k(\mathbf{w}_{t+1}^k) - f^k(\mathbf{w}_t^k) \leq -\psi ||\mathbf{g}_t^k||^2 \; \forall \; k \in [K],$$

with probability at least $1 - \delta$, where $\psi = \frac{\alpha^\star \beta}{M(1+\epsilon)}$.

Proof. See Appendix 5.6.1. □

We next use the result in Lemma 8 to prove linear convergence for the global function $f(\cdot)$. We first prove guarantees for the $L = 1$ case, where the models are communicated every iteration but the gradient is computed locally instead of globally contrary to previous results [51] (thus reducing two communication rounds per iteration). We then extend it to the general case of $L > 1$ and show that the updates converge at a sublinear rate in that case.

Theorem 8. *[$L = 1$ case] Suppose Assumptions 1-5 hold and the step-size α_t^k satisfies the line-search condition* (5.5)*. Also, let $0 < \delta < 1$, $0 < \epsilon, \epsilon_1 < 1/2$ be fixed constants and let $\Gamma = \max_{1 \leq i \leq n} \|\nabla f_i(.)\|$. Moreover, assume that the sample size for each worker satisfies $s \geq \frac{4B}{\kappa\epsilon^2} \log \frac{2dK}{\delta}$, where the samples are chosen without replacement. Then, with the LocalNewton updates, $\{\bar{\mathbf{w}}_t\}_{t \geq 0}$, from Algorithm 12 and $L = 1$, we obtain*

1. *If $s \gtrsim \frac{\Gamma^2}{\epsilon_1^2 G^2} \log(d/\delta)$ for $G = \min_k \|g^k(\bar{\mathbf{w}}_t)\|$, we get with probability at least $1 - 6K\delta$,*

$$f(\bar{\mathbf{w}}_{t+1}) - f(\mathbf{w}^*) \leq \rho_1 (f(\bar{\mathbf{w}}_t) - f(\mathbf{w}^*)).$$

2. *We obtain, with probability at least $1 - 6K\delta$,*

$$f(\bar{\mathbf{w}}_{t+1}) - f(\mathbf{w}^*) \leq \rho_2 (f(\bar{\mathbf{w}}_t) - f(\mathbf{w}^*)) + \eta \cdot \frac{\Gamma}{\kappa(1-\epsilon)},$$

where $\eta = \frac{1}{\sqrt{s}} \Gamma(1 + \sqrt{2\log(\frac{1}{\delta})})$.

[4] We introduce $\alpha^\star$ here to establish theoretical guarantees. In our empirical results, we use the Armijo backtracking line-search rule with $\alpha^\star = 1$ (e.g., see [149]) to find the right step-size.

Here $\rho_i = (1 - 2\kappa C_i)$*, for* $i = \{1, 2\}$*,* $C_1 = \frac{(1-\epsilon)\psi}{2} - \frac{\epsilon_1}{\kappa(1-\epsilon)}$*,* $C_2 = \frac{\psi(1-\epsilon)}{2}$*, and* $\psi = \frac{\alpha^\star \beta}{M(1+\epsilon)}$.

Proof. The proof is presented in Appendix 5.6.2. Here, we provide a sketch of the proof.

1. Due to the uniform sampling guarantee from Lemma 7, the strong-convexity and smoothness of the global function $f(\cdot)$ implies that the local function at the k-th worker, $f^k(\cdot)$, also satisfy similar properties. Using this, we can lower bound $f(\bar{\mathbf{w}}_t) - f(\bar{\mathbf{w}}_{t+1})$ in terms of $\frac{1}{K}\sum_{k=1}^K f^k(\mathbf{w}_t^k) - f^k(\mathbf{w}_{t+1}^k)$.
2. Apply Lemma 8 (that is, the result for standard Newton step) which says $f^k(\mathbf{w}_t^k) - f^k(\mathbf{w}_{t+1}^k) \geq \psi ||\mathbf{g}_t^k||^2 \; \forall \; k \in [K]$.
3. Using uniform sketching argument, local gradients $g^k(\bar{\mathbf{w}}_t)$ are close to global gradient $g(\bar{\mathbf{w}}_t)$.

□

Some remarks regarding the convergence guarantee in Theorem 8 are in order.

Remark 7. The above theorem implies that for $L = 1$, the convergence rate of LocalNewton is linear with high probability. Choosing $\delta = 1/\mathsf{poly}(K)$, we obtain the high probability as $1 - 1/\mathsf{poly}(K)$.

Remark 8. We have two different settings in the above theorem. The first setting implies that provided the local gradients $\{g^k(\bar{\mathbf{w}}_t)\}_{k=1}^K$ are large enough, and the amount of local data s is reasonably large, then the convergence is *purely* linear and does not suffer an error floor. This will typically happen in the earlier iterations of LocalNewton. Note that the gradients vanish as we get closer to the optimum, which is why the setting will eventually be violated. If this happens, we move to the next setting.

Remark 9. The second setting implies that, if the gradient condition and the restriction of s are violated, although the convergence rate of LocalNewton is still linear, the algorithm incurs an error floor. However, in this setting, the error floor is $\mathcal{O}(1/\sqrt{s})$, and hence it is quite small for sufficiently large sample-size, s, at each worker.

Assume all the workers initialize at $\bar{\mathbf{w}}_0$ and run LocalNewton with $L = 1$ for T iterations. Then, from Theorem 8, to reach within ξ of the optimal function value (that is, $f(\bar{\mathbf{w}}_T) - f(\bar{\mathbf{w}}^*) \leq \xi$), the number of iterations T is upper bounded by

$$T \leq \left(\log \frac{1}{\rho_1}\right) \log \frac{\xi}{f(\bar{\mathbf{w}}_0) - f(\bar{\mathbf{w}}^*)}$$

with probability $1 - 6K\delta$ for a sample size $s \geq \max\left\{\frac{4B}{\kappa\epsilon^2} \log \frac{2dKT}{\delta}, \frac{\Gamma^2}{\epsilon_1^2 G^2} \log \frac{dT}{\delta}\right\}$. (Note the increase in sample size s by a factor of T in the $\log(\cdot)$ due to a union bound). The fully synchronized second order method GIANT [51] also has similar linear quadratic convergence but it assumes that the gradients are synchronized in every iteration. We remove this assumption by tracking how far the iterate deviates when the gradients are computed locally, thereby cutting the communication costs in half while still showing linear convergence (within some error floor in the most general case).

We now prove convergence guarantees for the case when $L > 1$ in this algorithm.

Theorem 9 ($L \geq 1$ case)**.** *Suppose Assumptions 1-4 hold and step-size α_t^k solves the line-search condition* (5.5)*. Also, let $0 < \delta < 1$, $0 < \epsilon < 1/2$ be fixed constants and let $\Gamma = \max_{1 \leq i \leq n} \|\nabla f_i(.)\|$. Moreover, assume that the sample size for each worker satisfies $s \geq \frac{4B}{\kappa\epsilon^2} \log \frac{2dK}{\delta}$, where the samples are chosen without replacement. Then, the LocalNewton updates, $\{\bar{\mathbf{w}}_t\}_{t \geq 0}$, from Algorithm 12 and $L \geq 1$, with probability at least $1 - 6LK\delta$, satisfy*

$$f(\bar{\mathbf{w}}_{t+1}) - f(\bar{\mathbf{w}}_{t_0}) \leq -C \sum_{\tau=t_0}^{t} \left(\frac{1}{K} \sum_{k=1}^{K} \|\mathbf{g}_\tau^k\|^2 \right) + \eta \cdot \frac{L\Gamma}{\kappa(1-\epsilon)},$$

where $\eta = \frac{1}{\sqrt{s}} \Gamma(1 + \sqrt{2\log(\frac{1}{\delta})})$, $C = \psi - \frac{(M - \kappa(1-\epsilon)^2)}{2K\kappa^2(1-\epsilon)^2}$. where t_0 is the last iteration where the models were synced, $\psi = \frac{\alpha^\star \beta}{M(1+\epsilon)}$, and $C = \frac{\psi(1-\epsilon)^3}{2}$.

Proof. The general idea of the proof follows the proof for the easier case in Theorem 8 (i.e., when $L = 1$). The proof is presented in Appendix 5.6.3. □

Remark 10. The theorem shows that LocalNewton with high probability produces a descent direction, provided that the error floor is sufficiently small, i.e. for sufficiently large s (since η is proportional to $1/\sqrt{s}$). Observe that the convergence rate here is no longer linear. In other words, we are trading-off the rate of convergence for local iterations ($L > 1$).

Remark 11. Choosing $\delta = 1/\mathsf{poly}(K, L)$, we get that the theorem holds with probability at least $1 - 1/\mathsf{poly}(K, L)$. Note that this is not restrictive since the dependence on δ is logarithmic.

While the theoretical guarantees for $L > 1$ in Theorem 9 are not as strong as those for $L = 1$ in Theorem 8 (linear versus sublinear convergence), empirically we observe a fast rate of convergence even when $L > 1$ (see Section 5.5 for empirical results). Nevertheless, to the best of our knowledge, Theorem 9 is the first to show a descent guarantee for a distributed second-order method without synchronizing at every iteration. Obtaining a better rate of convergence for general L, with or without error floor, is an interesting and relevant future research direction.

5.5 Empirical Evaluation

In this section, we present an empirical evaluation of our approach when solving a large-scale logistic regression problem. We ran our experiments on AWS Lambda, a serverless computing platform which uses a high-latency cloud storage (AWS S3) to exchange data with the workers, using the PyWren [1] framework. We ran experiments on the real-world datasets described in Table 5.1 (obtained from LIBSVM [119]).

We compare the following distributed optimization schemes for the above datasets:
1. Local SGD [57]: The workers communicate their models once every epoch, where training on one epoch implies applying SGD (with mini-batch size one) over one pass of the dataset stored locally at the worker. The best step-size was obtained through hyperparameter tuning (see Table 5.2 for details).

Dataset	Training samples (n)	Features (d)	Testing samples
w8a	$48,000$	300	$15,000$
Covtype	$500,000$	2916	$81,000$
EPSILON	$400,000$	2000	$100,000$
a9a	$32,000$	123	$16,000$
ijcnn1	$49,000$	22	$91,000$

Table 5.1: Datasets considered for experiments in this chapter

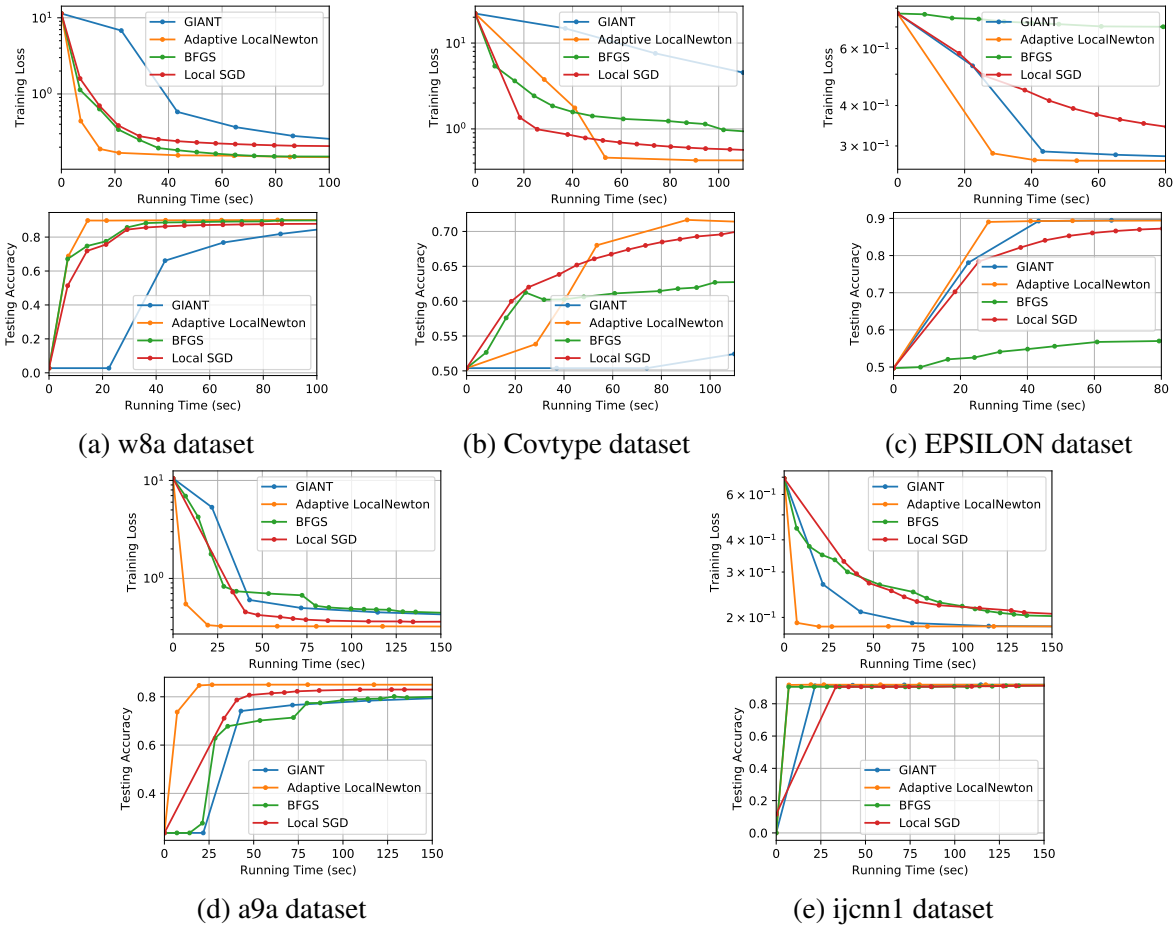
2. BFGS [164]: BFGS is a popular quasi-Newton method that estimates an approximate Hessian from the gradient information from previous iterations. The step-size was obtained using backtracking line-search. The best step-size was obtained through hyperparameter tuning (see Table 5.2 for details).
3. GIANT [51]: A state-of-the-art distributed second order algorithm proposed in [51]. The authors show that GIANT outperforms many popular schemes such as DANE, AGD, etc. The step-size was obtained using distributed line-search as described in [51].
4. Adaptive LocalNewton: For all the considered datasets, Adaptive LocalNewton gradually reduces L if the loss function stops decreasing, starting from $L = 3$ in the first round of communication. In general, during the later stages of optimization, it switches to GIANT owing to its convergence to the optimal solution when $\bar{\mathbf{w}}_t$ is sufficiently close to $\mathbf{w}^*$. The step-size was obtained using backtracking line-search locally at each worker as described in Algorithm 12.

For all the experiments presented in this chapter, we fixed the number of workers, K, to be 100. Hence, the number of samples per worker, $s = n/100$, for all datasets. The regularization parameter was chosen to be $\gamma = 1/n$. Note that there are several other schemes–such as AGD [157], DANE [48] and SVRG [165]–that have been proposed in the literature for communication-efficient optimization. However, most of these schemes have been shown to be outperformed by one of Local SGD, BFGS or GIANT, and hence, we do not perform the comparison again.

Hyperparameters for Local SGD and BFGS: In Table 5.2, we provide the step-sizes for local SGD and BFGS that were obtained through hyperparameter tuning, where $s = n/K$, n is the number of training examples in the dataset and $K = 100$.

Dataset	**Samples per worker (s)**	**Local SGD**	**BFGS**
w8a	480	$10/s$	100
Covtype	5000	$10/s$	1
EPSILON	4000	$500/s$	10
a9a	320	$10/s$	1
ijcnn1	490	$100/s$	10

Table 5.2: Step-sizes obtained using tuning for Local SGD and BFGS for several datasets



(a) w8a dataset (b) Covtype dataset (c) EPSILON dataset

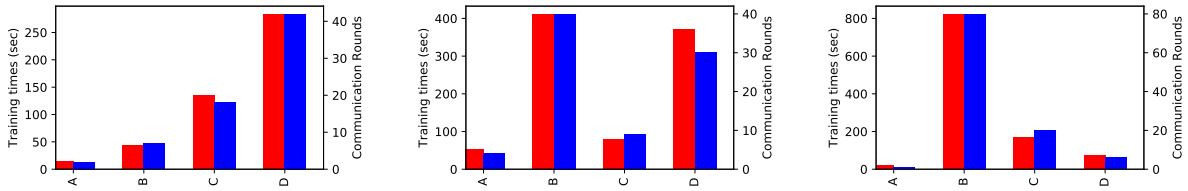
(d) a9a dataset (e) ijcnn1 dataset

Figure 5.3: Experiments on the different datasets from Table 5.1 on AWS Lambda. Both in terms of training loss and testing accuracy, Adaptive LocalNewton converges to the optimal value at least 50% faster than existing schemes.

Results: In Fig. 5.3, we plot the training loss and testing accuracy for w8a, covtype[5], EPSILON, a9a and ijcnn1 datasets. For all the datasets considered, Adaptive LocalNewton significantly outperforms its competitors in terms of time required to reach the same training loss (or testing accuracy).

In Fig. 5.4, we highlight the fact that runtime savings on AWS Lambda are a direct consequence of significantly fewer rounds of communication. Specifically, to reach the same training loss, we plot the training times and communication rounds as bar plots for three datasets, and we note that savings in communication rounds results in commensurate savings on end-to-end runtimes on AWS Lambda.

[5]The covtype dataset has $d = 54$ features and it does not perform well with logistic regression. Hence, we apply polynomial feature extension (using pairwise products) to increase the number of features to $d^2 = 2916$.



(a) w8a dataset (b) Covtype dataset (c) EPSILON dataset

Figure 5.4: Training times (red bars) and communication rounds (blue bars) required to reach the same training loss of $0.19, 0.65$ and 0.3 for the w8a, Covtype and EPSILON datasets, respectively, on AWS Lambda. Here, A: Adaptive LocalNewton, B: BFGS, C: Local SGD, D: GIANT.

5.6 Proofs

5.6.1 Auxiliary Lemmas and their Proofs

Here, we prove the auxiliary lemmas that are used in the main proofs of the paper. (For completeness, we restate the lemma statements).

Lemma 9. *Let $f(\cdot)$ satisfy assumptions 1-4 and $0 < \epsilon \leq 1/2$ and $\delta < 1$ be fixed constants. Then, if $s \geq \frac{4B}{\kappa\epsilon^2} \log \frac{2d}{\delta}$, the local Hessian at the k-th worker satisfies*

$$(1-\epsilon)\kappa \preccurlyeq \nabla^2 f^k(\mathbf{w}) = \mathbf{H}^k(\mathbf{w}) \preccurlyeq (1+\epsilon)M, \tag{5.7}$$

for all $\mathbf{w} \in \mathbb{R}^d$ and $k \in [K]$ with probability (w.p.) at least $1-\delta$.

Proof. At the k-th worker which samples $\mathcal{S}_k$ observations from $[n]$, the following is true by Matrix Chernoff (see Theorem 2.2 in Tropp (2011))

$$\mathbb{P}(\lambda_{\min}\left(\nabla^2 f^k(\mathbf{w})\right) \leq (1-\epsilon)\kappa) \leq \delta_1 = d\left[\frac{e^{-\epsilon}}{(1-\epsilon)^{1-\epsilon}}\right]^{s\kappa/B}, \tag{5.8}$$

$$\mathbb{P}(\lambda_{\max}\left(\nabla^2 f^k(\mathbf{w})\right) \geq (1+\epsilon)M) \leq \delta_2 = d\left[\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right]^{sM/B}. \tag{5.9}$$

Now, using the inequality $\log(1-\epsilon) \leq \frac{-\epsilon}{\sqrt{1-\epsilon}}$ for $0 \leq \epsilon < 1$, we get

$$\frac{e^{-\epsilon}}{(1-\epsilon)^{1-\epsilon}} \leq e^{-\epsilon+\epsilon\sqrt{1-\epsilon}}.$$

Further, utilizing the fact that $\sqrt{1-\epsilon} \leq \frac{1}{1+\epsilon/2}$, we get

$$e^{-\epsilon+\epsilon\sqrt{1-\epsilon}} \leq e^{\frac{-\epsilon^2}{1+\epsilon/2}} \leq e^{-\epsilon^2/4}.$$

Hence, we have $\delta_1 \leq de^{-s\kappa\epsilon^2/4B}$. Further, using the fact that $\log(1+\epsilon) \geq \epsilon - \epsilon^2/2$, we get

$$\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}} \leq e^{-\epsilon^2/2+\epsilon^3/2} \leq e^{-\epsilon^2/4},$$

where the last inequality follows from the fact that $\epsilon \leq 1/2$. Hence, $\delta_2 \leq de^{-sM\epsilon^2/4B}$. Thus, by union bound and subsequently using upper bounds on δ_1 and δ_2, we get

$$\begin{aligned}\mathbb{P}\left[(1-\epsilon)\kappa\mathbf{I} \preccurlyeq \nabla^2 f^k(\mathbf{w}) \preccurlyeq (1+\epsilon)M\mathbf{I}\right] &\geq 1-(\delta_1+\delta_2)\\ &\geq 1-(de^{-s\kappa\epsilon^2/4B}+de^{-sM\epsilon^2/4B})\\ &\geq 1-(2de^{-s\kappa\epsilon^2/4B}),\end{aligned}$$

where the last inequality follows from the fact that $\kappa \leq M$. Hence, the result follows by noting that

$$(1-\epsilon)\kappa\mathbf{I} \preccurlyeq \nabla^2 f^k(\mathbf{w}) \preccurlyeq (1+\epsilon)M\mathbf{I} \text{ w. p. at least } 1-\delta,$$

and requiring that $\delta \geq 2de^{-s\kappa\epsilon^2/4B}$ (or $s \geq \frac{4B}{\kappa\epsilon^2}\log\frac{2d}{\delta}$).

□

Lemma 10. *Let the function $f(\cdot)$ satisfy assumptions 1-3, and step-size α_t^k that solves the line-search condition in Eq. (5). Also, let $0<\epsilon\leq 1/2$ and $0<\delta<1$ be fixed constants. Moreover, let the sample size $s \geq \frac{4B}{\kappa\epsilon^2}\log\frac{2d}{\delta}$. Then, the LocalNewton update at the k-th worker satisfy*

$$f^k(\mathbf{w}_{t+1}^k) - f^k(\mathbf{w}_t^k) \leq -\psi||\mathbf{g}_t^k||^2 \;\forall\; k \in [K],$$

w.p. at least $1-\delta$, where $\psi = \frac{\alpha^\star\beta}{M(1+\epsilon)}$.

Proof. From Lemma 9, we know that $f^k(\cdot)$ is $M(1-\epsilon)$ smooth with probability $1-\delta$. M-smoothness of a function $g(\cdot)$ implies

$$g(\mathbf{y}) - g(\mathbf{x}) \leq (\mathbf{y}-\mathbf{x})^T\nabla g(\mathbf{x}) + \frac{M}{2}||\mathbf{y}-\mathbf{x}||^2 \;\forall\; \mathbf{x},\mathbf{y}\in\mathbb{R}^d. \tag{5.10}$$

Hence,

$$f^k(\mathbf{w}_t^k - \alpha\mathbf{p}_t^k) - f^k(\mathbf{w}_t^k) \leq (-\alpha\mathbf{p}_t^k)^T\mathbf{g}^k(\mathbf{w}_t^k) + \frac{M(1-\epsilon)}{2}\alpha^2||\mathbf{p}_t^k||^2. \tag{5.11}$$

The above inequality is satisfied for all $\alpha\in\mathbb{R}$. We know that α_t^k, the local step-size at worker k, satisfies the line-search constraint in Eq. (5). Thus, for $\alpha_t^k \in (0,1]$ to exist that satisfies the line-search condition, it is enough to find $\alpha>0$ that satisfies

$$-\alpha(\mathbf{p}_t^k)^T\mathbf{H}_t^k\mathbf{p}_t^k + \frac{M(1-\epsilon)}{2}\alpha^2\|\mathbf{p}_t^k\|^2 \leq -\alpha\beta(\mathbf{p}_t^k)^T\mathbf{H}_t^k\mathbf{p}_t^k, \tag{5.12}$$

where we have used the fact that $\mathbf{g}_t^k = \mathbf{H}_t^k\mathbf{p}_t^k$. Thus, α must satisfy

$$\frac{M(1-\epsilon)}{2}\alpha\|\mathbf{p}_t^k\|^2 \leq (1-\beta)(\mathbf{p}_t^k)^T\mathbf{H}_t^k\mathbf{p}_t^k. \tag{5.13}$$

Using lemma 9, we know that for sufficiently large sample-size at the k-th worker, we get

$$(1-\epsilon)\nabla^2 f(\mathbf{w}) \preceq \nabla^2 f^k(\mathbf{w}) \preceq (1+\epsilon)\nabla^2 f(\mathbf{w}) \tag{5.14}$$

with probability $1-\delta$. Also, by κ-strong convexity of $f(\cdot)$, we know that $\nabla^2 f(\mathbf{w}) \succeq \kappa \mathbf{I}$. Thus, the local line-search constraint is always satisfied for

$$\alpha \leq \frac{2(1-\beta)\kappa(1-\epsilon)}{M(1+\epsilon)}.$$

Hence, if we choose $\alpha^\star \leq \frac{2(1-\beta)\kappa(1-\epsilon)}{M(1+\epsilon)}$, or $\alpha^\star \leq \frac{\kappa(1-\beta)}{M}$ for $\epsilon < 1/2$, we are guaranteed to have the line-search condition from Eq. (5) satisfied with $\alpha_t^k = \alpha^\star$. This is satisfied by the line search equation in Eq. (5). Hence, from the line-search guarantee, we get

$$\begin{aligned} f^k(\mathbf{w}_{t+1}^k) - f^k(\mathbf{w}_t^k) &\leq -\alpha^\star \beta (\mathbf{p}_t^k)^T \mathbf{g}_t^k & (5.15)\\ &= \alpha^\star \beta (\mathbf{g}_t^k)^T (\mathbf{H}_t^k)^{-1} \mathbf{g}_t^k, & (5.16)\\ &\leq -\alpha^\star \beta \frac{1}{M(1+\epsilon)} \|\mathbf{g}_t^k\|^2, & (5.17) \end{aligned}$$

w.p. $1-\delta$. Here, the last inequality uses the fact that $f^k(\cdot)$ is $M(1+\epsilon)$–smooth, that is, $\mathbf{H}_t^k \preceq M(1+\epsilon)\mathbf{I}$. This proves the desired result. □

5.6.2 Proof of Theorem 8

The proofs for theorems in this paper use the auxiliary lemmas in Appendix 5.6.1.

Proof. The proof of the theorem is based on the following two high probability lower bounds:

Case 1:

$$f(\bar{\mathbf{w}}_t) - f(\bar{\mathbf{w}}_{t+1}) \geq C \|\mathbf{g}(\bar{\mathbf{w}}_t)\|^2, \tag{5.18}$$

where $C = \frac{\alpha^\star \beta (1-\epsilon)}{2M(1+\epsilon)}$ is a constant, and

Case 2

$$f(\bar{\mathbf{w}}_t) - f(\bar{\mathbf{w}}_{t+1}) \geq C_1 \|\mathbf{g}(\bar{\mathbf{w}}_t)\|^2 - \frac{\eta\Gamma}{\kappa(1-\epsilon)}, \tag{5.19}$$

where C_1 is a constant (> 0) and $\eta = (1 + \sqrt{2\log(\frac{1}{\delta})})\sqrt{\frac{1}{s}}\Gamma$.

We will prove the above result shortly, but let us complete the proof of the theorem assuming that Eq. (5.18) and Eq. (5.19) are true.

Case 1 (using Eq. (5.18)**)** Invoking the κ strong convexity of the the function f we have

$$f(\bar{\mathbf{w}}_t) - f(\mathbf{w}^*) \leq \frac{1}{2\kappa}\|\mathbf{g}(\bar{\mathbf{w}}_t)\|^2, \tag{5.20}$$

where $\bar{\mathbf{w}}^*$ is the unique global minimizer of the function f. Combining the last lower bound with equation (5.18) we obtain

$$f(\bar{\mathbf{w}}_{t+1}) - f(\bar{\mathbf{w}}_t) \leq (1 - 2\kappa C)(f(\bar{\mathbf{w}}_t) - f(\mathbf{w}^*)), \tag{5.21}$$

with probability $1 - \delta$. Also note that

$$1 > 1 - 2\kappa C = 1 - \frac{\kappa\alpha^\star\beta(1-\epsilon)}{M(1+\epsilon)} > 0,$$

where the last inequality uses the definition of $\alpha^\star$ from Eq. (6). The completes the proof of Theorem 3.2.

Case 2 (using Eq. (5.19)**)** Using the same steps as before, and using the condition of Eq. (5.19), we obtain Theorem 3.2.

It remains to prove the claim (5.18) and (5.19).

Proof of the claim (5.18)**:** Recall that for $L = 1$, we have

$$\mathbf{w}_{t+1}^k = \bar{\mathbf{w}}_t - \alpha_t^k \mathbf{p}_t^k, \quad \text{and} \quad \bar{\mathbf{w}}_{t+1} := \frac{1}{K}\sum_{k=1}^K \mathbf{w}_{t+1}^k = \bar{\mathbf{w}}_t - \frac{1}{K}\sum_{k=1}^K \alpha_t^k \mathbf{p}_t^k,$$

where the $\mathbf{p}_t^k = (\mathbf{H}_t^k)^{-1}\mathbf{g}_t^k, \mathbf{H}_t^k = (\mathbf{H}^k)^{-1}(\bar{\mathbf{w}}_t)$ and $\mathbf{g}_t^k = \mathbf{g}^k(\bar{\mathbf{w}}_t)$. Invoking the M-smoothness of the function $f(\cdot)$ we have

$$\begin{aligned}
f(\bar{\mathbf{w}}_t) - f(\bar{\mathbf{w}}_{t+1}) &\geq \frac{-M}{2K^2}\|\bar{\mathbf{w}}_t - \bar{\mathbf{w}}_{t+1}\|^2 + \langle \mathbf{g}(\bar{\mathbf{w}}_t), \bar{\mathbf{w}}_t - \bar{\mathbf{w}}_{t+1}\rangle \\
&\geq \frac{-M}{2K^2}\left\|\sum_{k=1}^K (\alpha_t^k)\mathbf{p}_t^k\right\|^2 + \langle \mathbf{g}(\bar{\mathbf{w}}_t), \frac{1}{K}\sum_{k=1}^K \alpha_t^k \mathbf{p}_t^k\rangle \\
&\overset{(i)}{\geq} \frac{-M}{2K}\sum_{k=1}^K (\alpha_t^k)^2\|\mathbf{p}_t^k\|^2 + \langle \mathbf{g}(\bar{\mathbf{w}}_t), \frac{1}{K}\sum_{k=1}^K \alpha_t^k \mathbf{p}_t^k\rangle \\
&= \frac{1}{K}\sum_{k=1}^K \left(\alpha_t^k (\mathbf{p}_t^k)^T \mathbf{g}(\bar{\mathbf{w}}_t) - \frac{M}{2}(\alpha_t^k)^2\|\mathbf{p}_t^k\|^2\right)
\end{aligned} \tag{5.22}$$

where the inequality (i) uses the following fact

$$\left\|\frac{1}{K}\sum_{k=1}^K \mathbf{a}^k\right\|^2 \leq \frac{1}{K}\sum_{k=1}^K \|\mathbf{a}^k\|^2, \tag{5.23}$$

for all vectors $\mathbf{a}^1, \mathbf{a}^2, \cdots, \mathbf{a}^K \in \mathbb{R}^d$.

We now complete the proof by using the following bound on the first term in Eq. (5.22). In particular, In the first case, we show that, for all $k \in [K]$ provided

$$s \gtrsim \left(\frac{\Gamma^2}{\epsilon_1^2 G^2} \log(d/\delta) \right),$$

and $\|\mathbf{g}^k(\bar{\mathbf{w}}_t)\| \geq G$, where $\epsilon_1 > 0$ (small number), we have

$$\alpha_t^k (\mathbf{p}_t^k)^T \mathbf{g}(\bar{\mathbf{w}}_t) \geq \left(\psi - \frac{\epsilon_1}{\kappa(1-\epsilon)} \right) \|\mathbf{g}_t^k\|^2 + \frac{\kappa(1-\epsilon)(\alpha_t^k)^2}{2} \|\mathbf{p}_t^k\|^2 \tag{5.24}$$

with probability at least $1 - 4\delta$.

Let us substitute Eq. (5.24) in equation (5.22), we get

$$\begin{aligned} f(\bar{\mathbf{w}}_t) - f(\bar{\mathbf{w}}_{t+1}) &\geq \frac{1}{K} \sum_{k=1}^{K} \left[\left(\psi - \frac{\epsilon_1}{\kappa(1-\epsilon)} \right) \|\mathbf{g}_t^k\|^2 - \frac{(M - \kappa(1-\epsilon))(\alpha_t^k)^2}{2} \|\mathbf{p}_t^k\|^2 \right] \\ &\geq \frac{1}{K} \sum_{k=1}^{K} \left[\left(\psi - \frac{\epsilon_1}{\kappa(1-\epsilon)} \right) \|\mathbf{g}_t^k\|^2 - \frac{(M - \kappa(1-\epsilon))(\alpha_t^k)^2}{2\kappa^2(1-\epsilon)^2} \|\mathbf{g}_t^k\|^2 \right] \end{aligned} \tag{5.25}$$

where the last inequality follows from the fact that the function f^k is $\kappa(1-\epsilon)$ strongly convex with probability $1 - \delta$, and thus

$$\|\mathbf{p}_t^k\|^2 := \|(\mathbf{H}_t^k)^{-1} \mathbf{g}_t^k\|_2^2 \leq \|(\mathbf{H}_t^k)^{-1}\|_2^2 \|\mathbf{g}_t^k\|^2 \leq \frac{1}{\kappa^2(1-\epsilon)^2} \|\mathbf{g}_t^k\|^2. \tag{5.26}$$

with probability $1 - \delta$. Now, using the upper bound on α_t^k, we have

$$\begin{aligned} f(\bar{\mathbf{w}}_t) - f(\bar{\mathbf{w}}_{t+1}) &\geq \frac{1}{K} \sum_{k=1}^{K} \left[\left(\psi - \frac{\epsilon_1}{\kappa(1-\epsilon)} \right) \|\mathbf{g}_t^k\|^2 - \frac{(M - \kappa(1-\epsilon)^2)}{2} \frac{\alpha^{\star 2}}{\kappa^2(1-\epsilon)^2} \|\mathbf{g}_t^k\|^2 \right] \\ &= \left(\psi - \frac{\epsilon_1}{\kappa(1-\epsilon)} - \frac{(M - \kappa(1-\epsilon)^2)}{2} \frac{\alpha^{\star 2}}{\kappa^2(1-\epsilon)^2} \right) \frac{1}{K} \sum_{k=1}^{K} \|\mathbf{g}_t^k\|^2 \\ &\geq C \frac{1}{K} \sum_{k=1}^{K} \|\mathbf{g}_t^k\|^2, \end{aligned} \tag{5.27}$$

with probability exceeding $1 - 6\delta$, where $C = \frac{(1-\epsilon)\psi}{2} - \frac{\epsilon_1}{\kappa(1-\epsilon)}$, and the last bound follows by substituting the value of α^* from equation (6) and using the fact that $0 < \epsilon < 1/2$. Moreover, using Eq. (5.23), we get

$$\|\mathbf{g}(\cdot)\|^2 \leq \frac{1}{K} \sum_{k=1}^{K} \|\mathbf{g}^k(\cdot)\|^2,$$

which prove Eq. (5.18).

It now remains to prove bound (5.24).

Proof of bound (5.24)**:** From the uniform subsampling property (similar to Lemma 9, see Appendix 5.6.6), we get

$$|(\mathbf{p}_t^k)^T \mathbf{g}(\bar{\mathbf{w}}_t) - (\mathbf{p}_t^k)^T \mathbf{g}^k(\bar{\mathbf{w}}_t)| \leq \epsilon_1 \|(\mathbf{p}_t^k)\| \|\mathbf{g}_k(\bar{\mathbf{w}}_t)\| \text{ w.p. } 1-\delta. \tag{5.28}$$

Thus,

$$(\mathbf{p}_t^k)^T \mathbf{g}(\bar{\mathbf{w}}_t) \geq (\mathbf{p}_t^k)^T \mathbf{g}_k(\bar{\mathbf{w}}_t) - \epsilon_1 \|(\mathbf{p}_t^k)\| \|\mathbf{g}_k(\bar{\mathbf{w}}_t)\| \tag{5.29}$$

w.p. $1-\delta$. Now, since the function f^k is $\kappa(1-\epsilon)$ strongly-convexity with probability $1-\delta$, we have the following bound w.p. at least $1-\delta$:

$$\alpha_t^k (\mathbf{p}_t^k)^T \mathbf{g}_t^k \geq (f^k(\bar{\mathbf{w}}_t) - f^k(\mathbf{w}_{t+1}^k)) + \frac{\kappa(1-\epsilon)}{2} (\alpha_t^k)^2 \|\mathbf{p}_t^k\|^2 \tag{5.30}$$

Combing the equations (5.29)-(5.30) and using Lemma 10 we have

$$\begin{aligned}
\alpha_t^k (\mathbf{p}_t^k)^T \mathbf{g}(\bar{\mathbf{w}}_t) &\geq (f^k(\bar{\mathbf{w}}_t) - f^k(\mathbf{w}_{t+1}^k)) + \frac{\kappa(1-\epsilon)}{2} (\alpha_t^k)^2 \|\mathbf{p}_t^k\|^2 - \epsilon_1 \|(\mathbf{p}_t^k)\| \|\mathbf{g}_k(\bar{\mathbf{w}}_t)\| \\
&\overset{(i)}{\geq} \psi \|\mathbf{g}_t^k\|^2 + \frac{\kappa(1-\epsilon)}{2} (\alpha_t^k)^2 \|\mathbf{p}_t^k\|^2 - \epsilon_1 \|(\mathbf{p}_t^k)\| \|\mathbf{g}_k(\bar{\mathbf{w}}_t)\| \\
&\overset{(ii)}{\geq} \psi \|\mathbf{g}_t^k\|^2 + \frac{\kappa(1-\epsilon)(\alpha_t^k)^2}{2} \|\mathbf{p}_t^k\|^2 - \frac{\epsilon_1}{\kappa(1-\epsilon)} \|\mathbf{g}_k(\bar{\mathbf{w}}_t)\|^2 \\
&= \left(\psi - \frac{\epsilon_1}{\kappa(1-\epsilon)}\right) \|\mathbf{g}_t^k\|^2 + \frac{\kappa(1-\epsilon)(\alpha_t^k)^2}{2} \|\mathbf{p}_t^k\|^2
\end{aligned}$$

with probability exceeding $1-4\delta$, where the inequality (i) follows from Lemma 10 and inequality (ii) follows from (5.26).

Note that the bound in (5.24) hold for all $k \in [K]$ with probability $1-\delta_1$ (thus, the sample size increases by a factor of K in the $\log(\cdot)$ term). This concludes the Case 1 of our proof. We now move to Case 2.

Proof of the claim (5.19)**:** We now continue with the same analysis and show the following

$$f(\bar{\mathbf{w}}_t) - f(\bar{\mathbf{w}}_{t+1}) \geq C_1 \frac{1}{K} \sum_{k=1}^{K} \|\mathbf{g}_t^k\|^2 - \frac{\eta\Gamma}{\kappa(1-\epsilon)}, \tag{5.31}$$

with probability at least $1-4\delta$.

In this case, we show that the requirement of a lower bound on $\|g^k(\bar{\mathbf{w}}_t)\|$ and s can be relaxed at the expense of getting hit by an error floor. In particular, we show that

$$\alpha_t^k (\mathbf{p}_t^k)^T \mathbf{g}(\bar{\mathbf{w}}_t) \geq \psi \|\mathbf{g}_t^k\|^2 + \frac{\kappa(1-\epsilon)(\alpha_t^k)^2}{2} \|\mathbf{p}_t^k\|^2 - \frac{\eta\Gamma}{\kappa(1-\epsilon)} \tag{5.32}$$

with probability at least $1-4\delta$, where $\eta = (1+\sqrt{2\log(\frac{1}{\delta})})\sqrt{\frac{1}{s}}\Gamma$. Substituting this yields the bound of Eq. (5.31).

Proof of bound Eq. (5.32) : From the uniform subsampling property (see Appendix 5.6.5), we get

$$|(\mathbf{p}_t^k)^T\mathbf{g}(\bar{\mathbf{w}}_t) - (\mathbf{p}_t^k)^T\mathbf{g}^k(\bar{\mathbf{w}}_t)| \leq \eta\|(\mathbf{p}_t^k)\| \text{ w.p. } 1-\delta. \tag{5.33}$$

where $\eta = (1+\sqrt{2\log(\frac{1}{\delta})})\sqrt{\frac{1}{s}}\Gamma$. Thus,

$$(\mathbf{p}_t^k)^T\mathbf{g}(\bar{\mathbf{w}}_t) \geq (\mathbf{p}_t^k)^T\mathbf{g}_k(\bar{\mathbf{w}}_t) - \eta\|(\mathbf{p}_t^k)\| \tag{5.34}$$

w.p. $1-\delta$. Now, since the function f^k is $\kappa(1-\epsilon)$ strongly-convexity with probability $1-\delta$, we have the following bound w.p. at least $1-\delta$:

$$\alpha_t^k(\mathbf{p}_t^k)^T\mathbf{g}_t^k \geq (f^k(\bar{\mathbf{w}}_t) - f^k(\mathbf{w}_{t+1}^k)) + \frac{\kappa(1-\epsilon)}{2}(\alpha_t^k)^2\|\mathbf{p}_t^k\|^2 \tag{5.35}$$

Combing the equations (5.34)-(5.35) and using Lemma 10 we have

$$\begin{aligned}
\alpha_t^k(\mathbf{p}_t^k)^T\mathbf{g}(\bar{\mathbf{w}}_t) &\geq (f^k(\bar{\mathbf{w}}_t) - f^k(\mathbf{w}_{t+1}^k)) + \frac{\kappa(1-\epsilon)}{2}(\alpha_t^k)^2\|\mathbf{p}_t^k\|^2 - \eta\|(\mathbf{p}_t^k)\| \\
&\overset{(i)}{\geq} \psi\|\mathbf{g}_t^k\|^2 + \frac{\kappa(1-\epsilon)}{2}(\alpha_t^k)^2\|\mathbf{p}_t^k\|^2 - \eta\|(\mathbf{p}_t^k)\| \\
&\overset{(ii)}{\geq} \psi\|\mathbf{g}_t^k\|^2 + \frac{\kappa(1-\epsilon)(\alpha_t^k)^2}{2}\|\mathbf{p}_t^k\|^2 - \frac{\eta}{\kappa(1-\epsilon)}\Gamma
\end{aligned}$$

with probability exceeding $1-4\delta$, where the inequality (i) follows from Lemma 10 and inequality (ii) follows from (5.26) and the fact that $\|\mathbf{g}^k(\bar{\mathbf{w}}_t)\| \leq \Gamma$.

□

5.6.3 Proof of Theorem 9

Proof. Recall from perturbed iterate analysis

$$\bar{\mathbf{w}}_{t+1} = \bar{\mathbf{w}}_{t_0} - \sum_{\tau=t_0}^{t}\bar{\mathbf{p}}_\tau, \tag{5.36}$$

where $\bar{\mathbf{p}}_\tau = \frac{1}{K}\sum_{k=1}^K \alpha_\tau^k\mathbf{p}_\tau^k$ is the average descent direction and $\mathbf{p}_\tau^k = (\mathbf{H}_\tau^k)^{-1}\mathbf{g}_\tau^k$ is the local descent direction at the k-th worker at time τ.

Similar to the proof of theorem 3.2, we next invoke the M-smoothness property of $f(\cdot)$ to get

$$\begin{aligned}
f(\bar{\mathbf{w}}_{t_0}) - f(\bar{\mathbf{w}}_{t+1}) &\geq \frac{-M}{2}\|\sum_{\tau=t_0}^{t}\bar{\mathbf{p}}_\tau\|^2 + \langle\mathbf{g}(\bar{\mathbf{w}}_{t_0}), \sum_{\tau=t_0}^{t}\bar{\mathbf{p}}_\tau\rangle \\
&= \frac{-M}{2}\|\frac{1}{K}\sum_{k=1}^K\sum_{\tau=t_0}^{t}\alpha_\tau^k\mathbf{p}_\tau^k\|^2 + \frac{1}{K}\sum_{k=1}^K\sum_{\tau=t_0}^{t}\langle\mathbf{g}(\bar{\mathbf{w}}_{t_0}), \alpha_\tau^k\mathbf{p}_\tau^k\rangle \\
&\geq \frac{-M}{2K}\sum_{k=1}^K\|\sum_{\tau=t_0}^{t}\alpha_\tau^k\mathbf{p}_\tau^k\|^2 + \frac{1}{K}\sum_{k=1}^K\sum_{\tau=t_0}^{t}\langle\mathbf{g}(\bar{\mathbf{w}}_{t_0}), \alpha_\tau^k\mathbf{p}_\tau^k\rangle,
\end{aligned} \tag{5.37}$$

where the last inequality uses the fact

$$\left(\sum_{k=1}^{K}\|\mathbf{a}_k\|\right)^2 \leq K\sum_{k=1}^{K}\|\mathbf{a}_k\|^2,\ \forall\ \mathbf{a}_k\in\mathbb{R}^d, k\in[K]. \tag{5.38}$$

Similarly, by $\kappa(1-\epsilon)$ strong-convexity of $f^k(\cdot)$, we get

$$f^k(\mathbf{w}_{t_0}^k) - f^k(\mathbf{w}_{t+1}^k) \leq \frac{-\kappa(1-\epsilon)}{2}\|\sum_{\tau=t_0}^{t}\alpha_\tau^k\mathbf{p}_t^k\|^2 + \langle\mathbf{g}_t^k, \sum_{\tau=t_0}^{t}\alpha_\tau^k\mathbf{p}_\tau^k\rangle, \tag{5.39}$$

with probability $1-\delta$. The above inequality, when averaged across k, becomes

$$\frac{1}{K}\sum_{k=1}^{K}\left(f^k(\mathbf{w}_{t_0}^k) - f^k(\mathbf{w}_{t+1}^k)\right) \leq \frac{-\kappa(1-\epsilon)}{2K}\sum_{k=1}^{K}\|\sum_{\tau=t_0}^{t}\alpha_\tau^k\mathbf{p}_t^k\|^2 + \frac{1}{K}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\langle\mathbf{g}_t^k, \alpha_\tau^k\mathbf{p}_\tau^k\rangle \tag{5.40}$$

Moreover, similar to Eq. (5.33), we get

$$|\mathbf{r}^T\mathbf{g}(\bar{\mathbf{w}}_t) - \mathbf{r}^T\mathbf{g}^k(\bar{\mathbf{w}}_t)| \leq \eta\|\mathbf{r}\| \text{ w.p. } 1-\delta. \tag{5.41}$$

where $\eta = (1+\sqrt{2\log(\frac{m}{\delta})})\sqrt{\frac{1}{s}}\Gamma$. Keeping $\mathbf{r} = \alpha_\tau^k\mathbf{p}_\tau^k$ and $\mathbf{w} = \bar{\mathbf{w}}_{t_0}$, we get

$$(\alpha_\tau^k\mathbf{p}_\tau^k)^T\mathbf{g}(\bar{\mathbf{w}}_{t_0}) \geq (\alpha_\tau^k\mathbf{p}_\tau^k)^T\mathbf{g}^k(\bar{\mathbf{w}}_{t_0}) - \eta\alpha_\tau^k\|\mathbf{p}_\tau^k\|, \tag{5.42}$$

w. p. $1-\delta$, where $\eta = (1+\sqrt{2\log(\frac{m}{\delta})})\sqrt{\frac{1}{s}}\Gamma$.

Now, after combining inequalities (5.37) and (5.40) using (5.42) to eliminate the terms $\frac{1}{K}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\langle\mathbf{g}(\bar{\mathbf{w}}_{t_0}), \alpha_\tau^k\mathbf{p}_\tau^k\rangle$ and $\frac{1}{K}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\langle\mathbf{g}^k(\bar{\mathbf{w}}_{t_0}), \alpha_\tau^k\mathbf{p}_\tau^k\rangle$, we get

$$\begin{aligned} f(\bar{\mathbf{w}}_{t_0}) - f(\bar{\mathbf{w}}_{t+1}) \geq \frac{1}{K}\sum_{k=1}^{K}(f^k(\bar{\mathbf{w}}_{t_0}^k) - f^k(\bar{\mathbf{w}}_{t+1}^k) - \frac{(M-\kappa(1-\epsilon))}{2K}\sum_{k=1}^{K}(\|\sum_{\tau=t_0}^{t}\alpha_\tau^k\mathbf{p}_t^k\|^2) \\ - \frac{1}{K}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\eta\alpha_\tau^k\|\mathbf{p}_\tau^k\|. \end{aligned} \tag{5.43}$$

Also, from Lemma 10, we have

$$f^k(\bar{\mathbf{w}}_{t_0}^k) - f^k(\bar{\mathbf{w}}_{t+1}^k) \geq \psi\sum_{\tau=t_0}^{t}\|\mathbf{g}_\tau^k\|^2. \tag{5.44}$$

Using above, we get

$$\begin{aligned} f(\bar{\mathbf{w}}_{t_0} - f(\bar{\mathbf{w}}_{t+1}) \geq \frac{1}{K}\psi\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\|\mathbf{g}_\tau^k\|^2 - \frac{(M-\kappa(1-\epsilon))}{2K}\sum_{k=1}^{K}(\|\sum_{\tau=t_0}^{t}\alpha_\tau^k\mathbf{p}_t^k\|^2) \\ - \frac{1}{K}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\eta\alpha_\tau^k\|\mathbf{p}_\tau^k\|. \end{aligned} \tag{5.45}$$

Using triangle inequality above, we get

$$f(\bar{\mathbf{w}}_{t_0}) - f(\bar{\mathbf{w}}_{t+1}) \geq \frac{1}{K}\psi \sum_{k=1}^{K}\sum_{\tau=t_0}^{t} \|\mathbf{g}_\tau^k\|^2 - \frac{(M-\kappa(1-\epsilon))}{2K}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}(\alpha_\tau^k)^2\|\mathbf{p}_\tau^k\|^2 - \frac{1}{K}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\eta\alpha_\tau^k\|\mathbf{p}_\tau^k\|. \tag{5.46}$$

Also, since $\alpha_t^k \leq 1$ and $\|\mathbf{p}_\tau^k\| \leq \frac{1}{\kappa(1-\epsilon)}\|\mathbf{g}_\tau^k\|$, we get

$$\begin{aligned} f(\bar{\mathbf{w}}_{t_0} - f(\bar{\mathbf{w}}_{t+1}) &\geq \frac{1}{K}\psi \sum_{k=1}^{K}\sum_{\tau=t_0}^{t} \|\mathbf{g}_\tau^k\|^2 - \frac{(M-\kappa(1-\epsilon))}{2K\kappa^2(1-\epsilon)^2}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\|\mathbf{g}_\tau^k\|^2 \\ &\quad - \frac{1}{K}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\frac{\eta}{\kappa(1-\epsilon)}\|\mathbf{g}_\tau^k\| \\ &= \frac{C}{K}\sum_{k=1}^{K}\sum_{\tau=t_0}^{t}\|\mathbf{g}_\tau^k\|^2 - \frac{\eta L\Gamma}{\kappa(1-\epsilon)} \end{aligned} \tag{5.47}$$

where $C = \psi - \frac{(M-\kappa(1-\epsilon))}{2K\kappa^2(1-\epsilon)^2}$, which proves the claim.

□

5.6.4 Concentration Inequalities: With and without Error Floor

Consider a vector $v \in \mathbb{R}^d$. We have defined the following: $\mathbf{g}(\bar{\mathbf{w}}_t) = \frac{1}{n}\sum_i \mathbf{g}_i(\bar{\mathbf{w}}_t)$ and $\mathbf{g}^k(\bar{\mathbf{w}}_t) = \frac{1}{s}\sum_{i\in\mathcal{S}}\mathbf{g}_i(\bar{\mathbf{w}}_t)$, where $\mathbf{g}_i$ denotes the local gradient in worker machine i, and $\mathcal{S}$ is the random set consisting data points for machine k. Let us do the calculation in two settings:

5.6.5 With error floor

Here we have the error floor. Note that having an error floor is not restrictive, if we go for the adaptive variation of the algorithm, where we run GIANT for the final iterations. Since GIANT has no error floor, the final accuracy won't be affected by the error floor obtained in the first few steps of the algorithm (check if this is true).

Lemma 11 (McDiarmid's Inequality)**.** *Let $X = X_1, \ldots, X_m$ be m independent random variables taking values from some set A, and assume that $f : A^m \to \mathbb{R}$ satisfies the following condition (bounded differences):*

$$\sup_{x_1,\ldots,x_m,\hat{x}_i} |f(x_i,\ldots,x_i,\ldots,x_m) - f(x_i,\ldots,\hat{x}_i,\ldots,x_m)| \leq c_i,$$

for all $i \in \{1,\ldots,m\}$. Then for any $\epsilon > 0$ we have

$$P\left[f(X_1,\ldots,X_m)-\mathbb{E}[f(X_1,\ldots,X_m)]\geq\epsilon\right]\leq\exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^m c_i^2}\right).$$

The property described in the following is useful for uniform row sampling matrix.

Let $\mathbf{S}\in\mathbb{R}^{n\times s}$ be any uniform sampling matrix, then for any matrix $\mathbf{B}=[\mathbf{b}_1,\ldots,\mathbf{b}_n]\in\mathbb{R}^{d\times n}$ with probability $1-\delta$ for any $\delta>0$ we have,

$$\|\frac{1}{n}\mathbf{BSS}^\top\mathbf{1}-\frac{1}{n}\mathbf{B1}\|\leq(1+\sqrt{2\log(\frac{1}{\delta})})\sqrt{\frac{1}{s}}\max_i\|\mathbf{b}_i\|, \tag{5.48}$$

where $\mathbf{1}$ is all ones vector.

Let us first see the justification of the above statement.The vector $\mathbf{B1}$ is the sum of column of the matrix $\mathbf{B}$ and $\mathbf{BSS}^\top\mathbf{1}$ is the sum of uniformly sampled and scaled column of the matrix $\mathbf{B}$ where the scaling factor is $\frac{1}{\sqrt{sp}}$ with $p=\frac{1}{n}$. If $(i_1,\ldots,i_s)$ is the set of sampled indices then $\mathbf{BSS}^\top\mathbf{1}=\sum_{k\in(i_1,\ldots,i_s)}\frac{1}{sp}\mathbf{b}_k$. Define the function $f(i_1,\ldots,i_s)=\|\frac{1}{n}\mathbf{BSS}^\top\mathbf{1}-\frac{1}{n}\mathbf{B1}\|$.

Now consider a sampled set $(i_1,\ldots,i_{j'},\ldots,i_s)$ with only one item (column) replaced then the bounded difference is

$$\begin{aligned}\Delta&=|f(i_1,\ldots,i_j,\ldots,i_s)-f(i_1,\ldots,i_{j'},\ldots,i_s)|\\&=|\frac{1}{n}\|\frac{1}{sp}\mathbf{b}_{i'_j}-\frac{1}{sp}\mathbf{b}_{i_j}\||\leq\frac{2}{s}\max_i\|\mathbf{b}_i\|.\end{aligned}$$

Now we have the expectation

$$\begin{aligned}&\mathbb{E}[\|\frac{1}{n}\mathbf{BSS}^\top\mathbf{1}-\frac{1}{n}\mathbf{B1}\|^2]\leq\frac{n}{sn^2}\sum_{i=1}^n\|\mathbf{b}_i\|^2=\frac{1}{s}\max_i\|\mathbf{b}_i\|^2\\\Rightarrow&\mathbb{E}[\|\frac{1}{n}\mathbf{BSS}^\top\mathbf{1}-\frac{1}{n}\mathbf{B1}\|]\leq\sqrt{\frac{1}{s}}\max_i\|\mathbf{b}_i\|.\end{aligned}$$

Using McDiarmid inequality (Lemma 11) we have

$$P\left[\|\frac{1}{n}\mathbf{BSS}^\top\mathbf{1}-\frac{1}{n}\mathbf{B1}\|\geq\sqrt{\frac{1}{s}}\max_i\|\mathbf{b}_i\|+t\right]\leq\exp\left(-\frac{2t^2}{s\Delta^2}\right).$$

Equating the probability with δ we have

$$\begin{aligned}&\exp(-\frac{2t^2}{s\Delta^2})=\delta\\\Rightarrow&t=\Delta\sqrt{\frac{s}{2}\log(\frac{1}{\delta})}=\max_i\|\mathbf{b}_i\|\sqrt{\frac{2}{s}\log(\frac{1}{\delta})}.\end{aligned}$$

Finally we have with probability $1-\delta$

$$\|\frac{1}{n}\mathbf{B}\mathbf{S}\mathbf{S}^\top\mathbf{1}-\frac{1}{n}\mathbf{B}\mathbf{1}\| \leq (1+\sqrt{2\log(\frac{1}{\delta})})\sqrt{\frac{1}{s}}\max_i \|\mathbf{b}_i\|,$$

and hence equation (5.48) is justified.

We now apply the above in distributed gradient estimation. For the k-th worker machine, we have

$$\|\frac{1}{n}\mathbf{B}\mathbf{S}_k\mathbf{S}_k^\top\mathbf{1}-\frac{1}{n}\mathbf{B}\mathbf{1}\| \leq (1+\sqrt{2\log(\frac{1}{\delta})})\sqrt{\frac{1}{s}}\max_i \|\mathbf{b}_i\|,$$

with probability $1-\delta$, which implies

$$\|\mathbf{g}^k(\bar{\mathbf{w}}_t)-\mathbf{g}(\bar{\mathbf{w}}_t)\| \leq (1+\sqrt{2\log(\frac{1}{\delta})})\sqrt{\frac{1}{s}}\Gamma,$$

with probability at least $1-\delta$ provided $\|\mathbf{g}_i(\bar{\mathbf{w}}_t)\| \leq \Gamma$ for all $i \in [m]$

Writing, $\eta = (1+\sqrt{2\log(\frac{1}{\delta})})\sqrt{\frac{1}{s}}L$, we succinctly write

$$|\langle v, \mathbf{g}^k(\bar{\mathbf{w}}_t)-\mathbf{g}(\bar{\mathbf{w}}_t)\rangle| \leq \|v\|\|\mathbf{g}^k(\bar{\mathbf{w}}_t)-\mathbf{g}(\bar{\mathbf{w}}_t)\| \leq \eta\|v\|$$

with probability at least $1-\delta$, where $\eta = \mathcal{O}(1/\sqrt{s})$ is small.

5.6.6 Without error floor

In this section, we analyze the same quantity using vector Bernstein inequality. Intuitively, we show that unless $\mathbf{g}(\bar{\mathbf{w}}_t)$ is too small, we can overcome the error floor shown in the previous calculation. In particular, we assume that

$$\|\mathbf{g}^k(\bar{\mathbf{w}}_t)\| \geq G.$$

The idea here is to use the vector Bernstein inequality. Using the notation of Appendix 5.6.5, $\mathbf{g}^k(\bar{\mathbf{w}}_t) = \frac{1}{n}\mathbf{B}\mathbf{S}\mathbf{S}^\top\mathbf{1}$, where $\mathbf{S}$ is appropriately defined sampling matrix. Also $\mathbf{g}(\bar{\mathbf{w}}_t) = \frac{1}{n}\mathbf{B}\mathbf{1}$. For the k-th machine,

$$\mathbf{g}^k(\bar{\mathbf{w}}_t) = \frac{1}{s}\sum_{i\in\mathcal{S}}\mathbf{g}_i(\bar{\mathbf{w}}_t),$$

and so,

$$\mathbf{g}^k(\bar{\mathbf{w}}_t)-\mathbf{g}(\bar{\mathbf{w}}_t) = \frac{1}{s}\sum_{i\in\mathcal{S}}(\mathbf{g}_i(\bar{\mathbf{w}}_t)-\mathbf{g}(\bar{\mathbf{w}}_t)),$$

with $|\mathcal{S}| = s$. We also have $\|\mathbf{g}_i(\bar{\mathbf{w}}_t) - \mathbf{g}(\bar{\mathbf{w}}_t)\| \leq \Gamma + \Gamma = 2\Gamma$, and $\mathbb{E}\|\mathbf{g}_i(\bar{\mathbf{w}}_t) - \mathbf{g}(\bar{\mathbf{w}}_t)\|^2 \leq 4\Gamma^2$. Using vector Bernstein inequality with $t = \epsilon_1\|\mathbf{g}^k\|$, we obtain

$$\mathbb{P}\left(\|\mathbf{g}^k(\bar{\mathbf{w}}_t) - \mathbf{g}(\bar{\mathbf{w}}_t)\| \geq \epsilon_1\|\mathbf{g}^k(\bar{\mathbf{w}}_t)\|\right) \leq d\exp(-s\frac{\epsilon_1^2\|\mathbf{g}^k\|^2}{32\Gamma^2} + 1/4) \leq d\exp(-s\frac{\epsilon_1^2 G^2}{32L^2} + 1/4).$$

So, as long as

$$G^2 = \Omega\left(\frac{\Gamma^2}{\epsilon_1^2 s}\log(d/\delta)\right),$$

or,

$$s \gtrsim \left(\frac{\Gamma^2}{\epsilon_1^2 G^2}\log(d/\delta)\right),$$

we have,

$$|\langle v, \mathbf{g}^k(\bar{\mathbf{w}}_t) - \mathbf{g}(\bar{\mathbf{w}}_t)\rangle| \leq \|v\|\|\mathbf{g}^k(\bar{\mathbf{w}}_t) - \mathbf{g}(\bar{\mathbf{w}}_t)\| \leq \epsilon_1\|v\|\|\mathbf{g}^k\|$$

with probability at least $1 - \delta$.

5.7 Conclusion

The practicality of second-order optimization methods has been questioned since naive ways to implement them require large compute and power storage to work with the Hessian. However, in the last few decades, trends such as Moore's law have made computation faster and memory cheaper, while improvements in communication costs have been at best marginal. These trends, combined with a flurry of efficient but approximate algorithms [6], [42], [44], [45], have revived interest in second-order methods. In this paper, we identify and concretize the role that second-order methods—combined with local optimization algorithms—can play in reducing the communication costs during distributed training, in particular in serverless environments. Since second-order information has recently been used to develop state-of-the-art methods for deep neural networks with extremely large model sizes [66], [67], [160], [161], we expect that methods such as ours will play a significant role in motivating and designing next-generation communication-efficient algorithms for fast distributed training of machine learning models.

Chapter 6

BEAR: Feature Selection in Sublinear Memory

In this chapter, we consider the training of high-dimensional models that do not fit into the machine memory. We capitalize on the observation that for many such models, only a subset of features (sub-linear in model dimension) are relevant for prediction. To this end, we develop a second-order ultra-high dimensional feature selection algorithm, called BEAR, which stores the *second-order stochastic gradients* in the celebrated Broyden Fletcher Goldfarb Shannon (BFGS) algorithm using a Count Sketch, a sublinear memory data structure from the streaming literature.

6.1 Introduction

Consider a data set comprising n data points $(\boldsymbol{\theta}_i)_{i=1}^n = (\mathbf{x}_i, y_i)_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^p$ denotes the data vectors representing p features and $(y_i)_{i=1}^n$ denote the corresponding labels. Feature selection seeks to select a small subset of the features of size $k \ll p$ that best models the relationship between $\mathbf{x}_i$ and y_i. In this chapter, we consider the feature selection problem in ultra-high dimensional settings where dense feature vectors in $\mathbb{R}^p$ cannot fit in the working memory of the computer because of the sheer dimensionality of the problem (p). Such problems have become increasingly important in networking, biology, and streaming applications. In biology, it is common to represent a DNA sequence comprised of four nucleotides A, T, C, G, as well as 11 wild-card characters in the FASTQ format [166] using the frequency of sub-sequences of length k, called k-mers, with $k \geq 12$ [167], [168]. A feature vector of size $15^{k=12}$ with floating-point numbers requires more than a petabyte of memory to store. This is simply larger than the memory capacity of the computers today. In streaming, the memory budget of the local edge computing devices is extremely small compared to the dimension of the data streams [169]. In both scenarios, it is critical to select a subset of the features that are most predictive of the outputs with *sublinear memory* cost in the dimensionality of the data.

Recently, first-order stochastic gradient descent (SGD) algorithms [69], [70] have been developed which extend the ideas in feature hashing (FH) [68] to feature selection. Instead of explicitly

storing the feature vectors, these algorithms store a low-dimensional sketch of the features in a data structure called Count Sketch [71], originated from the streaming literature. Count Sketch preserves the weights of the top-k features (i.e., the heavy hitters) in sublinear memory with high probability. This high probability guarantee, however, depends on the energy of the non-top-k coordinates in the SGD algorithm. In particular, the noise components of the gradients, which normally average out in the regular stochastic optimization, accumulate in the non-top-k coordinates of Count Sketch. This unwanted sketched noise increases the probability of collision in Count Sketch, deteriorates the quality of the recovered features, and results in poor memory-accuracy trade-offs. This is a critical problem since the only class of optimization algorithms that operates in such ultra-high dimensions does not select high-quality features when the memory budget is small.

We propose a novel optimization scheme to solve this critical problem in sketching. We improve the quality of the sketched gradients and correct for the unwanted collisions in the sketched domain using the information from the second-derivative of the loss function. Second-order methods have recently gained increasing attention in machine learning for their faster convergence [170], less reliance on the step size parameter [171], and their superior communication-computation trade-off in distributed processing [66]. Here, we uncover another key advantage of second-order optimization in improving memory-accuracy trade-off in sketching models trained on ultra-high dimensional data sets. We develop a second-order sublinear memory algorithm which finds high-quality features by *limiting the probability of extra collisions* due to the stochastic noise in Count Sketch. The contributions of the chapter are as follows:

Algorithm. We develop BEAR which, to the best of our knowledge, is the first quasi-Newton-type algorithm that achieves a sublinear memory cost in the dimension of the data. BEAR stores the product of the inverse Hessian and the gradient in the Broyden–Fletcher–Goldfarb–Shannon (BFGS) algorithm using a sublinear memory Count Sketch. BEAR updates Count Sketch in time quadratic in the sparsity of the data by operating only on the features that are active in each minibatch.[1]

Theory. We theoretically demonstrate that BEAR maintains the $\mathcal{O}(1/t)$ global convergence rate of the online version of BFGS algorithms [172] in t iterations. We show that the convergence rate is retained as we go from the ambient domain to the sketched domain. The analysis employs the Johnson–Lindenstrauss (JL) lemma over the projection operator in Count Sketch. In practice, we demonstrate that BEAR converges faster than the first-order feature selection algorithms, although improving convergence time is not the main focus of this work.

Simulations. We did extensive controlled sparse recovery simulations with data points drawn from the normal distribution. We demonstrate that, given a fixed memory budget to store the weights, BEAR recovers the ground truth features with a large phase transition gap — an important statistical performance metric from the compressive sensing literature. We show that BEAR's performance is highly consistent across a large range of values for the step size parameter because of the second-order nature of the algorithm.

Experiments. In real-world, ultra-high dimensional data sets from genomics, natural language processing, and networking, we demonstrate that BEAR requires $10 - 1000\times$ less memory space to achieve the same classification accuracy as the first-order methods. Moreover, BEAR achieves

[1]Codes are available at https://github.com/BEAR-algorithm/BEAR

$10-20\%$ higher classification accuracy given the same memory budget to store Count Sketch and selects more interpretable features in ultra-high dimensions using a personal laptop-size machine. Importantly, our results show an increase in the performance gap between the first- and second-order methods as the memory budget to store the model parameters decrease, which highlights the important advantages of second-order optimization in storing the sketched stochastic gradient vectors with a lower collision rate.

6.2 Review: Count Sketch

Count Sketch is a data structure which is originated from the streaming literature [71]. Its primary application is to approximately count the number of occurrences of a very large number (p) of objects in sublinear memory, when only the frequency of the most recurring elements (i.e., the heavy hitters) are of interest. Instead of storing a counter for all the p objects, Count Sketch *linearly* projects the count values using d independent random hash functions into a $m \ll p$ dimensional. Count Sketch keeps a matrix of counters (or bins) $\mathcal{S}$ of size $m \sim \mathcal{O}(\log p)$. The algorithm uses d random hash functions $h_j : \forall j \in \{1,\ 2, ...,\ d\}$ to map p-dimensional vectors to m/d bins, that is, $h_j : \{1,\ 2, ...,\ p\} \rightarrow \{1,\ 2, ...,\ m/d\}$. For any row j of sketch $\mathcal{S}$, component i of the vector is hashed into bin $\mathcal{S}(j, h_j(i))$. In addition to h_j, Count Sketch uses d random sign functions to map the components of the vectors randomly to $\{+1,\ -1\}$, that is, $s_j : \{1,\ 2, ...,\ p\} \rightarrow \{+1, -1\}$.

Count Sketch supports two operations: ADD(item i, increment Δ) and QUERY(item i). The ADD operation updates the sketch with any observed increment. More formally, for an increment Δ to an item i, the sketch is updated by adding $s_j(i)\Delta$ to the cell $\mathcal{S}(j, h_j(i))\ \forall j \in \{1,\ 2, ...,\ d\}$. The QUERY operation returns an estimate for component i, the median of all the d different associated counters. Count Sketch provides the following bound in recovering the top-k coordinates of the feature vector $\mathbf{z} \in \mathbb{R}^p$:

Theorem 10. *[71] Count Sketch finds approximate top-k coordinates z_i with $\pm\varepsilon\|\mathbf{z}\|_2$ error, with probability at least $1-\delta$, in space $\mathcal{O}(log(\frac{p}{\delta})(k + \frac{\|\mathbf{z}^{tail}\|_2^2}{(\varepsilon\zeta)^2}))$, where $\|\mathbf{z}^{tail}\|_2^2 = \sum_{i \notin top-k} z_i^2$ is the energy of the non-top-k coordinates and ζ is the k^{th} largest value in* $\mathbf{z}$.

Count Sketch recovers the top-k coordinates with a memory cost that grows only logarithmic with the dimension of the data p; and naturally, it requires the energy of the non-top-k coordinates to be sufficiently small. This is the property that we leverage in this chapter in order to improve feature selection accuracy in ultra-high dimensions.

6.3 Stochastic Sketching for Feature Selection

We first elaborate on how can we perform feature selection using Count Sketch. Recall that feature selection seeks to select a small subset of the features that best models the relationship between $\mathbf{x}_i$ and y_i. This relationship is captured using a sparse feature vector $\boldsymbol{\beta}^* \in \mathbb{R}^p$ that minimizes a given loss function $f(\boldsymbol{\beta}, \boldsymbol{\theta}) : \mathbb{R}^p \rightarrow \mathbb{R}$ using the optimization problem $\min_{\boldsymbol{\beta}} \mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{\beta}, \boldsymbol{\theta})]$, where

$\boldsymbol{\theta} \in \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_n\}$ denotes a data point in a data set of size n. This problem is solved using the empirical risk minimization

$$\boldsymbol{\beta}^* \coloneqq \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} f(\boldsymbol{\beta}, \boldsymbol{\theta}_i), \tag{6.1}$$

using the SGD algorithm which produces the updates $\boldsymbol{\beta}_{t+1} \coloneqq \boldsymbol{\beta}_t + \eta_t \mathrm{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ at iteration t, where η_t is the step size, the minibatch $\boldsymbol{\Theta}_t = \{\boldsymbol{\theta}_{t1}, \boldsymbol{\theta}_{t2}, \ldots, \boldsymbol{\theta}_{tb}\}$ contains b independent samples from the data, and $\mathrm{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t) = \sum_{i=1}^{b} \nabla_{\boldsymbol{\beta}_t} f(\boldsymbol{\beta}_t, \boldsymbol{\theta}_{ti})$ is the stochastic gradient of the instantaneous loss function $f(\boldsymbol{\beta}, \boldsymbol{\Theta}_t)$. In this chapter, we are interested in the setting where the dense feature vector $\boldsymbol{\beta}_t$ of size p cannot be stored in the memory of a computer. The most common approach in machine learning when dealing with such high dimensional problem is to project the data points (i.e., the features) into a lower dimensional space. Feature hashing (FH) is one of the most popular algorithms [68] which uses a universal hash function to project the features. While FH is ideal for prediction, it is not suited for feature selection; that is, the original important features cannot be recovered from the hashed ones.

The reason to stay hopeful in recovering the important features using sublinear memory is that the feature vector $\boldsymbol{\beta}^*$ is typically sparse in ultra-high dimensions. However, while the final feature vector is sparse, $\boldsymbol{\beta}_t$ becomes dense in the intermediate iterations of the algorithm. The workaround is to store a sketch of the intermediate non-sparse $\boldsymbol{\beta}_t$ via a low dimensional sketched vector $\boldsymbol{\beta}_t^s \in \mathbb{R}^m$ (with $m \ll p$) such that the important features are still recoverable. This results in the following sketched optimization steps $\boldsymbol{\beta}_{t+1}^s \coloneqq \boldsymbol{\beta}_t^s + \eta_t \mathbf{g}^s(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$, where $\mathbf{g}^s(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ is the sketched gradient vector. To enable the recovery of the important features from the sketched features the weights $\boldsymbol{\beta}_t^s \in \mathbb{R}^m$ can be stored in Count Sketch [69], [71]. Count Sketch preserves the information of the top-k elements (i.e., the heavy hitters) with high probability as long as the energy of the non-top-k coefficients is small (see Theorem 10) . The noise term $\mathbf{g}^s(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ in the SGD algorithm, however, contributes to the energy of the non-top-k coefficients. This is a critical problem since, unlike SGD in the ambient dimension, this spurious sketched noise does not cancel out until it becomes so large that it shows up in the top-k coordinates in Count Sketch. As a result, a large fraction of the memory in Count Sketch will be wasted to store the sketched noise term in the non-top-k coordinates, which results in poor memory-accuracy trade offs in first-order methods.

Algorithm 14: Limited-memory BFGS

Input: $\mathrm{g}(\hat{\boldsymbol{\beta}}_t, \boldsymbol{\Theta}_t)$ and $\{\mathbf{s}_i, \mathbf{r}_i\}_{i=t-\tau+1}^{t}$
$\rho_t = \frac{1}{\mathbf{r}_t^T \mathbf{s}_t}$.
$\mathbf{q}_t = \mathrm{g}(\hat{\boldsymbol{\beta}}_t, \boldsymbol{\Theta}_t)$,
for $i = t$ to $t - \tau + 1$ $\alpha_i = \rho_i \mathbf{s}_i^T \mathbf{q}_i$,
$\mathbf{q}_{i-1} = \mathbf{q}_i - \alpha_i \mathbf{r}_i$. $\mathbf{z}_{t-\tau} = \frac{\mathbf{r}_t^T \mathbf{s}_t}{\mathbf{r}_t^T \mathbf{r}_t} \mathbf{q}_{t-\tau}$, for $i = t - \tau + 1$ to t $\gamma_i = \rho_i \mathbf{r}_i^T \mathbf{z}_i$.
$\mathbf{z}_i = \mathbf{z}_{i-1} + \mathbf{s}_i(\alpha_i - \gamma_i)$. **Return:** $\mathbf{z}_t$

6.4 Challenges of Second-order Sketching in Ultra-High Dimension

In this chapter, we propose to develop a second-order optimization algorithm for feature selection to reduce the effect of collisions while sketching SGD into lower dimensions. Recall that the stochastic second-order Newton's method produces the updates $\beta_{t+1} \coloneqq \beta_t + \eta_t \mathbf{B}_t^{-1} \mathbf{g}(\beta_t, \mathbf{\Theta}_t)$, where $\mathbf{B}_t = \nabla^2_{\boldsymbol{\beta}_t} f(\boldsymbol{\beta}_t, \mathbf{\Theta}_t) \in \mathbb{R}^{p \times p}$ is the instantaneous Hessian matrix at iteration t computed over the minibatch $\mathbf{\Theta}_t$. There are three main challenges in sketching these updates for ultra-high dimensional feature selection: First, computing the Hessian and finding the matrix inverse is computationally hard as the matrix inverse operation is going to have at least cubic computational complexity in the problem dimension p. Second, even if we assume that the Hessian is diagonal (e.g., in AdaHessian [67]), storing the diagonal elements will have linear memory cost in p, which we can not afford in ultra-high dimension. Third, sketching the Hessian matrix directly is not possible using Count Sketch since the linear increments are happening over the gradients and not the Hessian matrix in SGD. Therefore, sketching the step Newton's method is highly a nontrivial problem.

Here, we build on recent works in fast second-order optimization. Quasi-Newton's methods such as the BFGS algorithm reduce the time complexity of Newton's method using an iterative update of the Hessian as a function of the variations in the gradients $\mathbf{r}_t = \mathbf{g}(\boldsymbol{\beta}_{t+1}, \mathbf{\Theta}) - \mathbf{g}(\boldsymbol{\beta}_t, \mathbf{\Theta})$ and the feature vectors $\mathbf{s}_t = \boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}_t$ which ensures that the Hessian satisfies the so-called secant equation $\mathbf{B}_{t+1}\mathbf{s}_t = \mathbf{r}_t$. While BFGS avoids the heavy computational cost involved in the matrix inversion, it still has a quadratic memory requirement. The limited-memory BFGS (LBFGS) algorithm reduces the memory requirement of the BFGS algorithm from quadratic to linear by estimating the product of the inverse Hessian and the gradient vector $\mathbf{z}_t = \mathbf{B}_t^{-1}\mathbf{g}(\boldsymbol{\beta}_t, \mathbf{\Theta}_t)$ without explicitly storing the Hessian [173]. This makes use of the difference vectors $\mathbf{r}_t$ and $\mathbf{s}_t$ from the last τ iteration of the algorithm (Alg. 14). Recently, an online version of the LBFGS algorithm (oLBFGS) [172] has also been developed with global convergence guarantees. However, all these quasi-Newton's algorithms fail to run on ultra-high dimensional data sets due to their linear memory requirement. How can we attain the benefits of second-order optimization for sketching in sublinear memory?

6.5 The BEAR Algorithm

BEAR estimates the second-derivative of the loss function from sketched features. Instead of explicitly storing the product of the inverse Hessian and the gradient (done in oLBFGS), BEAR maintains a Count Sketch $\mathcal{S}$ to store the feature weights in sublinear memory and time quadratic in the sparsity of the input data.

As detailed in Alg. 15, BEAR first initializes Count Sketch with zero weights and a top-k heap to store the top-k features. In every iteration, it samples b independent data points $\mathbf{\Theta}_t$ and identifies the active set $\mathcal{A}_t$, that is, the features that are present in $\mathbf{\Theta}_t$. It then queries Count Sketch to retrieve the feature weights that are in the intersection of the active set $\mathcal{A}_t$ and the top-k heap and set the weights for the rest of the features to zero. Next, it computes the stochastic gradient $\mathbf{g}(\boldsymbol{\beta}_t, \mathbf{\Theta}_t)$

Algorithm 15: BEAR

1 **Initialize**: $t = 0$, Count Sketch $\boldsymbol{\beta}^s_{t=0} = 0$, top-$k$ heap. **while** *stopping criteria not satisfied* **do**
2 Sample b independent data points in a minibatch $\boldsymbol{\Theta}_t = \{\boldsymbol{\theta}_{t1}, \ldots, \boldsymbol{\theta}_{tb}\}$.
3 Find the active set $\mathcal{A}_t$.
4 QUERY the feature weights in $\mathcal{A}_t \cap$ top-k from Count Sketch $\boldsymbol{\beta}_t = query(\boldsymbol{\beta}^s_t)$.
5 Compute stochastic gradient $\mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$.
6 Compute the descent direction with Alg. 14
7 $\mathbf{z}_t$ = LBFGS($\mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$, $\{\mathbf{s}_i, \mathbf{r}_i\}^t_{i=t-\tau+1}$).
8 ADD the sketch of $\mathbf{z}_t$ at the active set $\hat{\mathbf{z}}_t = {\mathbf{z}_t}^{\mathcal{A}_t}$ to Count Sketch
9 $\boldsymbol{\beta}^s_{t+1} \coloneqq \boldsymbol{\beta}^s_t - \eta_t \hat{\mathbf{z}}^s_t$.
10 QUERY the features weights in $\mathcal{A}_t \cap$ top-k from Count Sketch $\boldsymbol{\beta}_{t+1} = query(\boldsymbol{\beta}^s_{t+1})$.
11 Compute stochastic gradient $\mathbf{g}(\boldsymbol{\beta}_{t+1}, \boldsymbol{\Theta}_t)$.
12 Set $\mathbf{s}_{t+1} = \boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}_t$, and $\mathbf{r}_{t+1} = \mathbf{g}(\boldsymbol{\beta}_{t+1}, \boldsymbol{\Theta}_t) - \mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$.
13 Update the top-k heap.
14 $t = t + 1$.
15 **Return:** The top-k heavy-hitters in Count Sketch.

and uses it along with the difference vectors $\mathbf{r}_i$ and $\mathbf{s}_i$ from the last τ iterations to find the descent direction $\mathbf{z}_t$ using the LBFGS algorithm detailed in Alg. 14. Then, it adds the sketch of the descent direction $\mathbf{z}_t$ *only at the features in the active set* $\hat{\mathbf{z}}_t = {\mathbf{z}_t}^{\mathcal{A}_t}$ to Count Sketch. BEAR queries Count Sketch for the second time in order to update the difference vector $\mathbf{r}_{t+1} = \mathbf{g}(\boldsymbol{\beta}_{t+1}, \boldsymbol{\Theta}_t) - \mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$ and uses the sketch vector $\hat{\mathbf{z}}_t$ to set $\mathbf{s}_{t+1}$. The difference vector $\mathbf{r}_{t+1}$ captures the changes in the gradient vector as the content of Count Sketch change *over a fixed minibatch* $\boldsymbol{\Theta}_t$ [172]. Finally, BEAR updates the top-k heap with the active set $\mathcal{A}_t$ and moves on to the next iteration until the convergence criteria is met. To update the top-k heap, BEAR scans the features that have been changed in Count Sketch over the past iteration. If those features are already in the heap, it updates the values of those elements, and if the features are new, it inserts the new elements into the heap with a worst-case time complexity that grows logarithmic with the number of features k. In the rare scenario, where the intersection of the active set $\mathcal{A}_t$ and the top-k heap is empty the gradient can still be non-zero and can change the feature weightings.

The most time-costly step of BEAR, which computes the descent direction (step 5), is quadratic time in the sparsity of the data $|\mathcal{A}_t|$. Table 6.1 summarizes the worst-case memory complexity of the vectors involved in BEAR. The dominant term is Count Sketch $\boldsymbol{\beta}^s_t$ for which the memory complexity in terms of the top-k value are established in Theorem 10. The memory requirement to store the auxiliary vector $\mathbf{z}_t$ is only a constant τ times larger than the size of the active set which is negligible in streaming data-sparse features compared to the size of Count Sketch (see section 6.7).

Convergence Analysis. We now analyze the convergence of the algorithm. Consider the following standard assumptions [172] on the instantaneous functions $f(\boldsymbol{\beta}, \boldsymbol{\Theta})$ to prove the convergence of the BEAR algorithm: 1) The instantaneous objective functions are twice differentiable with the

$\boldsymbol{\beta}_t$	$\mathbf{s}_t$	$\mathbf{r}_t$	$\mathbf{z}_t$	$\boldsymbol{\beta}_t^s$	$\mathbf{g}(\boldsymbol{\beta}_t, \boldsymbol{\Theta}_t)$
k	$2\lvert\mathcal{A}_t\rvert$	$2\lvert\mathcal{A}_t\rvert$	$2\tau\lvert\mathcal{A}_t\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}_t\rvert$

Table 6.1: Memory cost of the vectors in BEAR.

instantaneous Hessian being positive definite, that is, the eigenvalues of the instantaneous Hessian satisfy $M_1\mathbf{I} \preceq \nabla^2_{\boldsymbol{\beta}} f(\boldsymbol{\beta}, \boldsymbol{\Theta}) \preceq M_2\mathbf{I}$, for some $0 < M_1 < M_2$. 2) The norm of the gradient of the instantaneous functions is bounded for all $\boldsymbol{\beta}$, that is, $\mathbb{E}_{\boldsymbol{\Theta}}[\|\mathbf{g}(\boldsymbol{\beta}, \boldsymbol{\Theta})\|^2 \mid \boldsymbol{\beta}] \leq S^2$. 3) The step sizes η_t are square-summable. More specifically, $\sum_{t=0}^{\infty} \eta_t = \infty$ and $\sum_{t=0}^{\infty} \eta_t^2 < \infty$. We prove the following theorem.

Theorem 11. *Let $f(\cdot)$ and the step sizes η_t satisfy the assumptions above. Let the size of Count Sketch be $m = \theta(\varepsilon^{-2}\log 1/\delta)$ with number of hashes $d = \theta(\varepsilon^{-1}\log 1/\delta)$ for $\varepsilon, \delta > 0$. Then, the Euclidean distance between updates $\boldsymbol{\beta}_t^s$ in the BEAR algorithm and the sketch of the solution of problem* (6.1) *converges to zero with probability $1 - \delta$, that is,*

$$\mathbb{P}(\lim_{t\to\infty} \|\boldsymbol{\beta}_t^s - \boldsymbol{\beta}^{s*}\|^2 = 0) = 1 - \delta, \tag{6.2}$$

where the probability is over the random realizations of random samples $\{\boldsymbol{\Theta}_t\}_{t=0}^{\infty}$. Furthermore, for the specific step size $\eta_t = \eta_0/(t + T_0)$ for some constants η_0 and T_0, the model parameters at iteration t satisfy

$$\mathbb{E}_{\boldsymbol{\Theta}}[f(\boldsymbol{\beta}_t^s, \boldsymbol{\Theta}) - \mathbb{E}[f(\boldsymbol{\beta}^{s*}, \boldsymbol{\Theta})] \leq \frac{C_0}{T_0 + t}, \tag{6.3}$$

with probability $1 - \delta$. Here, C_0 is a constant depending on the parameters of the sketching scheme, the above assumptions, and the objective function.

The proof makes use of the Johnson-Lindenstrauss (JL) lemma in projecting the second-order gradients in Count Sketch [174] which we defer to the Proofs section. For sufficiently sparse solutions the convergence in the ambient domain follows from convergence in the sketched domain (i.e., Theorem (11)) and the Count Sketch guarantee (i.e., Theorem (10)):

Corollary 2. *Let $\pi(\cdot)$ be a permutation on $\{1, 2, \cdots, p\}$ such that $\beta^*_{\pi(1)} \geq \beta^*_{\pi(2)} \geq \cdots \beta^*_{\pi(p)}$, where $\boldsymbol{\beta}^* = [\beta_1^*, \beta_2^*, \cdots, \beta_p^*]$ is the optimal solution to* (6.1)*. Also, let*

$$m = \max\left[\mathcal{O}\left(log(\frac{2p}{\delta})\left(k + \frac{\|\boldsymbol{\beta}^{*tail}\|_2^2}{(\varepsilon\zeta)^2}\right)\right), \theta\left(\frac{1}{\varepsilon^2}\log\frac{2}{\delta}\right)\right] \tag{6.4}$$

and number of hashes $d = \theta(\varepsilon^{-1}\log 2/\delta)$ where $\varepsilon, \delta > 0$, $\|\boldsymbol{\beta}^{*tail}\|_2^2 = \sum_{i=k+1}^{p}(\beta^*_{\pi(i)})^2$ and $\zeta = \beta^*_{\pi(k)}$. Then,

$$|\beta^*_{\pi(i)} - \beta_{t\pi(i)}| \leq \epsilon\|\boldsymbol{\beta}^*\|_2 \quad \textit{for all } i \in \{1, 2, \cdots, k\} \textit{ with probability } 1 - \delta, \tag{6.5}$$

where $\boldsymbol{\beta}_t = [\beta_{t1}, \beta_{t2}, \cdots, \beta_{tp}]$ is the output of the BEAR algorithm.

This completes the convergence proof of BEAR in sublinear memory in the ambient space.

6.6 Simulations

We have conducted sparse recovery simulations to evaluate the performance BEAR compared to the first-order feature selection algorithm in ultra-high dimension MISSION [69]. The synthetic simulations described in this section have ground truth features, so we can assess the algorithms in a more controlled environment and compare the results using a variant of the phase transition plot from the compressive sensing literature [175]. We also show the results of the full Newton's method version of our BEAR algorithm where we compute the Hessian matrix rather than its BFGS approximation (this algorithm cannot operate in large-scale settings). The same hash table (hash functions and random seeds) and step sizes are used for BEAR and MISSION. Hyperparameter search is performed to select the value of the step sizes in both algorithms. The entries of the data vectors $\mathbf{x}_i$ are sampled from an i.i.d. Gaussian distribution with zero mean and unit variance. The output labels y_i are set using a linear forward model $y_i = \mathbf{x}_i\boldsymbol{\beta}^*$, where $\boldsymbol{\beta}^*$ is a k-sparse ground truth feature vector. The indices of the support (i.e., the non-zero entries) and the weights of the non-zero entries in $\boldsymbol{\beta}^*$ are drawn uniformly at random respectively from the sets $[1, p]$ and $[0.8, 1.2]$ and MSE is used as the loss function. The same experiment is repeated 200 times with different realization of the data vectors $\mathbf{x}$. Convergence at iteration t is reached when the norm of the gradient drops below 10^{-7} consistently in all the algorithms. The algorithms are compared in terms of the accuracy in selecting the ground truth features as well as the sensitivity of the algorithms to the choice of the value of the step size.

Feature Selection Accuracy. The task is to select $k = 8$ features in a data set with $n = 900$ rows (data points) and $p = 1000$ columns (features). The size of Count Sketch is varied from 10% to 60% of the total memory required to store a $p = 1000$ dimensional feature vector. This ratio, that is the ratio of data dimension p to Count Sketch size, is called the compression factor. For each value of the compression factor, the experiment is repeated 200 times. Fig. 6.1A shows the fraction of iterations in which the algorithms find *all* the ground truth features correctly, that is, the probability of success. Fig. 6.1B illustrates the same results in terms of the average ℓ_2-norm of the error of the recovered feature vectors $\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2$. BEAR significantly outperforms MISSION in terms of the probability of success and the average ℓ_2-norm error. The gap is more pronounced in higher compression factors; given a compression factor of 3, MISSION has almost no power in predicting the correct features while the BEAR and Newton's methods achieve a 0.5 probability of success. Fig. 6.1A and B further suggest that the performance gap between BEAR and it's exact Hessian counterpart is small showing that the oLBFGS makes a good approximation to the Hessian in terms of the selected features.

Sensitivity to Step Size. The experimental setup is similar to the previous section except the Sketch size is fixed and step size varies. The experiment is repeated 200 times while varying the values of the step size η ranging from 10^{-7} to 10^{-1} and the probability of success is reported. Count Sketch of size 150×3 is used for both MISSION and BEAR. Fig. 6.1C illustrates the probability of success for BEAR and MISSION as a function of the step size. The plot shows that BEAR is fairly agnostic and MISSION is dependent on the choice of the step size η. MISSION's accuracy peaks around $\eta = 10^{-4}$ and sharply drops as η deviates from this value. BEAR's lower-dependence on step size is ideal for streaming settings where the statistics of the data might change over time

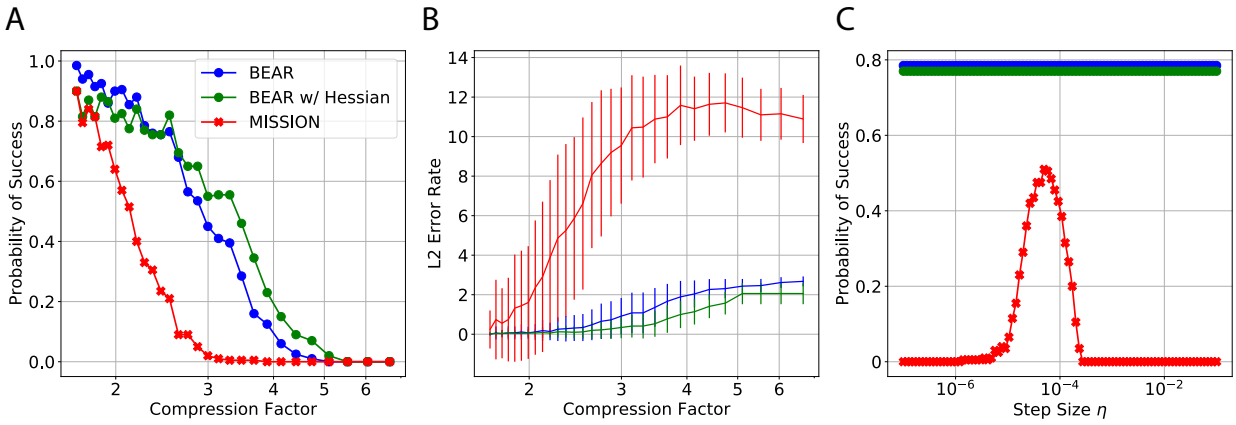


Figure 6.1: Feature selection experiments on $p = 1000$-dimensional synthetic data sets with entries drawn from normal distribution. A) Probability of success in recovering all features correctly as a function of compression factor. B) Recovery error rate in terms of the ℓ_2-norm. C) Probability of success as a function of the value of the step size (compression factor $= 2.22$).

and there is not enough time and memory budget to do step size selection.

6.7 Experiments

We designed the experiment in a way to answer the following questions:

- Does BEAR outperform MISSION in terms of classification accuracy? In particular, how does the performance gap between the algorithms change as a function of the memory allocated for Count Sketch?
- How does BEAR perform on real-world large-scale data sets ($p >$ 50 million)?
- How does BEAR perform in terms of classification accuracy compared to FH?
- How does changing the number of top-k features affect the accuracy of the feature selection algorithms?
- What is the convergence behaviour of BEAR when the memory budget is small?
- What is the run time of BEAR compared to MISSION?

We compare the performance of these baseline algorithms with BEAR:

1. **Stochastic Gradient Descent (SGD)**: For data sets with sufficiently small dimension and size to be able to train a classifier on our laptop machine, we perform the vanilla SGD algorithm (with $\mathcal{O}(p)$ memory).

Data set	Dim (p)	#Train (n)	#Test	Size	#Act.
RCV1	47,236	20,242	677,399	1.2GB	73
Webspam	16,609,143	280,000	70,000	25GB	3730
DNA	16,777,216	600,000	600,000	1.5GB	89
KDD 2012	54,686,452	119,705,032	29,934,073	22GB	12

Table 6.2: Summary of the real-world data sets.

2. **oLBFGS**: Similar to SGD, for the data sets that the dimension and size allows to train a classifier on our laptop machine, we perform the vanilla oLBFGS algorithm (neither SGD nor the oLBFGS techniques do feature selection or model compression).

3. **Feature Hashing (FH)**: FH [68] is a standard algorithm to do prediction (classification) in large-scale machine learning problems. FH hashes the data features into a lower dimensional space *before* the training process and is not a feature selection algorithm.

4. **MISSION**: As mentioned earlier, MISSION is a first-order optimization algorithm for feature selection which sketches the noisy stochastic gradients into Count Sketch.

Performance Metrics. The algorithms are assessed in terms of the following performance metrics:

1. **Classification accuracy**: Once the algorithms converge, the performance of the algorithms in terms of classification accuracy are compared, i.e., the fraction of test samples that are classified to correct classes.

2. **Area under the ROC curve (AUC)**: For the data sets that the class distribution are highly skewed the area under the ROC curve (AUC) is reported instead of the classification accuracy. In these data sets, the class probabilities are taken as the output of the classifiers.

3. **Compression factor (CF)**: The compression factor is defined as the dimension of the data set p divided by the size of Count Sketch m. For multi-class classification problems, m is the total memory of all the Count Sketches used for all the classes. A higher compression factor means a smaller memory budget is allocated to store the model parameters. SGD and oLBFGS have a compression factor of one.

4. **Run time**: The run time of the algorithms to converge in minutes.

Real-World Data sets. The key statistics of the data sets used in the chapter are tabulated in Table 6.2 including the dimension of the data set (p), number of training data (n), number of test data, total size of the data set, and the average number of active (non-zero) features per data point. All the data is analyzed in the Vowpal Wabbit format.

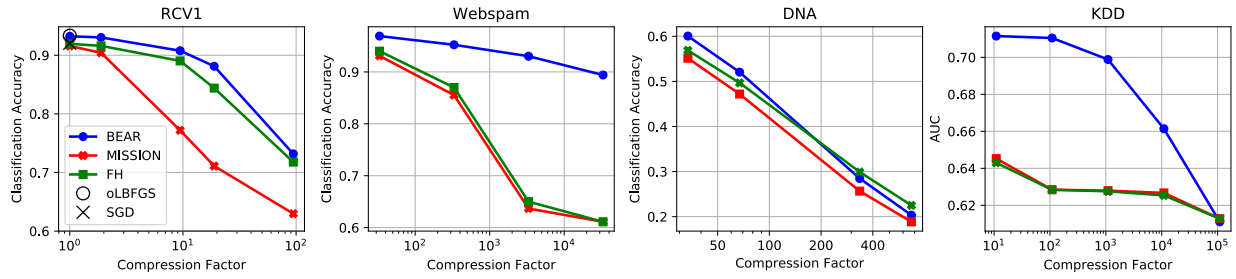


Figure 6.2: Classification performance as a function of the compression factor in real-world data sets.

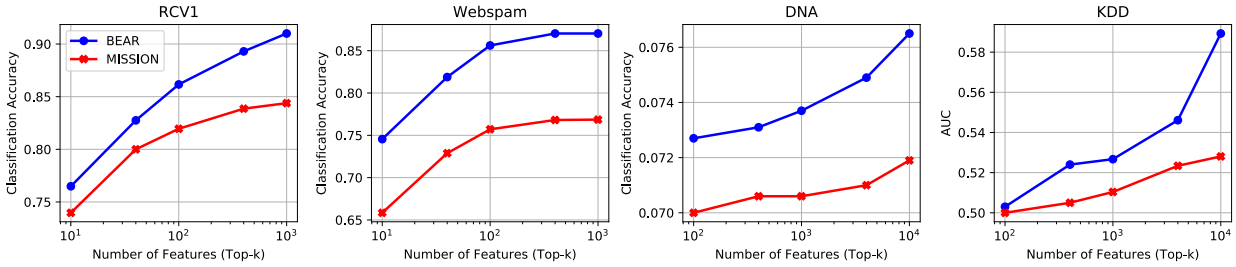


Figure 6.3: Classification performance as a function of the number of the top-k features.

1) **RCV1: Reuters Corpus Volume I.** RCV1 is an archive of manually categorized news wire stories made available by Reuters, Ltd. for research purposes. The negative class label includes Corporate/Industrial/Economics topics and positive class labels includes Government/Social/Markets topics (see [176]). The data set is fairly balanced between the two classes.

2) **Webspam: Web Spam Classification.** Web spam refers to Web pages that are created to manipulate search engines and Web users. The data set is a large collection of annotated spam/nonspam hosts labeled by a group of volunteers (see [177]). It is slightly class-imbalanced with 60% samples from class 1.

3) **DNA: Metagenomics.** A data set that we dub "DNA" from metagenomics. Metagenomics studies the composition of microbial samples collected from various environments (for example human gut) by sequencing the DNA of the living organisms in the sample. The data set comprises of short DNA sequences which are sampled from a set of 15 DNA sequences of bacterial genomes. The task is to train a classifier to label the DNA sequences with their corresponding bacteria. DNA sequences are encoded using their constituent sub-sequences called K-mers (see [167]). The training and test data have an equal number of samples for each class. A naive guessing strategy achieves a classification accuracy of 0.06.

4) **KDD Cup 2012: Click-Through Rate Prediction.** A key idea in search advertising is to predict the click-through rate (pCTR) of ads, as the economic model behind search advertising requires pCTR values to rank ads and to price clicks. The KDD Cup 2012 data set comprises training instances derived from session logs of the Tencent proprietary search engine (see [178]). The data set is highly class-imbalanced with 96% samples from class 1 (click).

Multi-class Extension. For the multi-class classification problems stated above, we developed a multi-class version of the BEAR algorithm. In the multi-class problem one natural assumption is that there are separate subsets of features that are most predictive for each class. Our multi-class BEAR algorithm accommodates for this by maintaining a separate Count Sketch and heap to store the top-k features associated with each class. The total memory complexity of the algorithm grows linearly with the number of classes. For a fair comparison, we use the exact same multi-class Count Sketch extension for MISSION. We have also implemented the single Count Sketch version of BEAR, however, since the multi-class Count Sketch extension performs better for feature selection we report the results of the former in our experiments.

Experimental Setup. MurmurHash3 with 32-bit hash values is used to implement the hash functions in MISSION, BEAR, and FH. The algorithms are trained in a streaming fashion using the cross entropy loss. The algorithms are run for a single epoch so that each algorithm sees a data point once on average. The size of the minibatches and the step size are kept consistent across the algorithms. The constant $\tau = 5$ in BEAR however the results are consistent across a large range of values for τ. Both in BEAR and MISSION a Count Sketch with 5 rows (hash functions) is used. The lower dimensional embedding size of FH is set equal to the total size of Count Sketch in BEAR. The experiments are performed on a single laptop machine - 2.4 GHz Quad-Core Intel Core i5 with 16 GB of RAM. We chose an edge device as opposed to a computing server for our real-world experiments to showcase the applicability of BEAR in a resource constrained environment.

Result I) Classification Performance vs. Compression Factor. We assess the classification performance of BEAR compared to the baseline algorithms for different compression factors in Fig. 6.2. All the active features in the test data are used at the inference step for a fair comparison with FH. BEAR's classification performance is consistently better than MISSION and FH across all the data sets over a wide range of compression factors while showing a hysteresis behaviour: the performance gap increases as the compression factor grows until Count Sketch is too small to yield any prediction power. The classification performance of all algorithms degrades with larger compression factors, which is expected since lower Count Sketch sizes increase the probability of collisions in both BEAR and MISSION. The degradation, however, impacts MISSION significantly more that BEAR. In particular, BEAR's performance stays relatively robust for compression rates in the range of $1 - 10$ in RCV1, $1 - 1000$ in Webspam, and $10 - 100$ in KDD, while the classification performance of MISSION drops rapidly. The increasing performance gap between BEAR and MISSON with compression factor highlights the unique advantage of BEAR in storing the second-order steps in Count Sketch and lowering the probability of collisions. Note that this performance gap is less pronounced in the DNA data set while the general trend still follows the other data sets. This is because the DNA data set has 15 balanced classes and its K-mer features have relatively more distributed information content compared to the features in the other data sets, which poses a harder feature selection task for the algorithms.

Result II) Classification Performance vs. Top-k Features. We assess the performance of BEAR in terms of the classification accuracy against the number of selected top-k features. The compression factors are fixed to 10, 330, 330, and 1100 respectively for the data sets in Fig. 6.3. SGD, oLBFGS, and FH cannot select features, therefore, they are not included in this analysis.

BEAR	manage	entrepreneur	colombian	decade	oppress
MISSION	peach	week	nora	demand	incomplete

Table 6.3: Examples of the features selected in RCV1.

data set (CF)	RCV1 (95)	Webs (332)	DNA (22)	KDD (10^3)
BEAR	0.1	5	26	25
MISSION	0.3	19	55	33

Table 6.4: Overall run time comparison (minutes).

The plots shows that BEAR selects features that are better in terms of prediction accuracy for a wide range of values of k. The gap grows for larger k. We analyzed the selected features in RCV1 for which a proper documentation of the features is publicly available (unlike the other data sets). Some of the selected features are shared among the algorithms, e.g., "shareholder", "nigh", and "company", which can be attributed to the Markets, Social, and Industrial subjects, respectively. Other terms, however, are uniquely chosen by one of the algorithms as tabulated in Table 6.3. Compared to BEAR, the terms selected by MISSION are less frequent (e.g., "peach") and do not discriminate between the subject classes (e.g., "incomplete").

Result III) Run Time. We compare the overall run time of BEAR with MISSION in Table 6.4. BEAR is significantly faster than MISSION consistently in all the data sets; BEAR makes a better use of the data by estimating the curvature of the loss function and converges faster.

6.8 Proofs

Theorem 2. Before stating the proof, for more clarity, we will reiterate the problem setup and our assumptions from the chapter here. We are interested in solving the following problem using BEAR

$$\boldsymbol{\beta}^* \coloneqq \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}}\, f(\boldsymbol{\beta}, \boldsymbol{\Theta}) = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}}\, \frac{1}{T}\sum_{t=1}^{T} f(\boldsymbol{\beta}, \boldsymbol{\theta}_t) = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}}\, \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{X}_t\boldsymbol{\beta}, \mathbf{y}_t), \tag{6.6}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_T\}$ and $\boldsymbol{\theta}_t = (\mathbf{X}_t, \mathbf{y}_t)\ \forall\ t \in [1, T]$. We make the following standard assumptions [172]:

1. The instantaneous objective functions, $f(\cdot)$, in Eq. (6.6) are twice differentiable with the instantaneous Hessian being positive definite. That is, the eigenvalues of the instantaneous Hessian satisfy

$$M_1\mathbf{I} \preceq \nabla^2_{\boldsymbol{\beta}} f(\boldsymbol{\beta}, \boldsymbol{\Theta}) \preceq M_2\mathbf{I}, \tag{6.7}$$

for some $0 < M_1 \leq M_2$.

2. The norm of the gradient of the instantaneous functions $f(\cdot)$ in Eq. 6.6 is bounded for all $\boldsymbol{\beta}$, that is

$$\mathbb{E}_{\boldsymbol{\Theta}}[\|\mathbf{g}(\boldsymbol{\beta}, \boldsymbol{\Theta})\|^2 \mid \boldsymbol{\beta}] \leq S^2. \tag{6.8}$$

3. The step-sizes η_t are square-summable. More specifically,

$$\sum_{t=0}^{\infty} \eta_t = \infty \text{ and } \sum_{t=0}^{\infty} \eta_t^2 < \infty. \tag{6.9}$$

Lemma 12. *The solution of problem in* (6.6) *using BEAR (or its first-order variant MISSION) is equivalent to the solution of the following problem in the sketched domain,*

$$\boldsymbol{\beta}^{s*} \coloneqq \underset{\boldsymbol{\beta}^s \in \mathbb{R}^m}{\operatorname{argmin}} \frac{1}{T} \sum_{t=1}^{T} f(\mathbf{X}_t \mathbf{S} \boldsymbol{\beta}^s, \mathbf{y}_t), \tag{6.10}$$

where multiplication by $\mathbf{S} \in \mathbb{R}^{p \times m}$ is the linear projection operator in Count Sketch and $\boldsymbol{\beta}^s \in \mathbb{R}^m$ is the projected model parameters.

Proof. Let the update for online gradient descent for the original problem in Eq. (6.6) be given by

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta_t \nabla f(\mathbf{X} \boldsymbol{\beta}_t, \mathbf{y}_t) \tag{6.11}$$

For BEAR/MISSION type algorithms, the model parameters are stored in a Count Sketch based hash table. The compressed vector can be represented by an affine transformation as $\boldsymbol{\beta}_t^s = \mathbf{S}^T \boldsymbol{\beta}_t$, where $\mathbf{S} \in \mathbb{R}^{p \times m}$ is the Count Sketch matrix [174]. While updating the model, the indices corresponding to the non-zero values in the gradient (the oLBFGS update in case of BEAR) are updated by querying Count Sketch. For Count Sketch with mean query operator, the update for MISSION can be written as

$$\boldsymbol{\beta}_{t+1}^s = \boldsymbol{\beta}_t^s - \eta_t \mathbf{S}^T \nabla f(\mathbf{X} \mathcal{Q}(\boldsymbol{\beta}_t^s), \mathbf{y}_t) \tag{6.12}$$

where $\mathcal{Q}(\cdot)$ is the query function and $\boldsymbol{\beta}_t^s = \mathbf{S}^T \boldsymbol{\beta}_t$ is the sketched model parameter vector. When the query is the mean operator, the $\mathcal{Q}(\cdot)$ is the affine transformation $\mathcal{Q}(x) = \mathbf{S}\mathbf{x}$ for any $\mathbf{x} \in \mathbb{R}^m$ [41], [174]. Thus, the MISSION update equation is given by

$$\boldsymbol{\beta}_{t+1}^s = \boldsymbol{\beta}_t^s - \eta_t \mathbf{S}^T \nabla f(\mathbf{X}\mathbf{S} \boldsymbol{\beta}_t^s, \mathbf{y}_t). \tag{6.13}$$

The gradient for the problem in Eq. (6.10) is given by $\nabla f_{\boldsymbol{\beta}^s}(\cdot) = \mathbf{S}^T \nabla f(\cdot)$. Hence, its online gradient descent update is the same as MISSION's update in Eq. (6.13). Since BEAR is a second-order variant of MISSION, it attempts to solve the same problem as MISSION. Next, we show that it indeed solves the problem with high probability at a linear convergence rate. □

Now, to show that BEAR converges to $\boldsymbol{\beta}^{s*}$, we first need to show that the problem in Eq. (6.10) also satisfies the assumptions in Eq. (6.7) and (6.8) (albeit with different constants). Then, we can invoke the convergence guarantees for oLBFGS from [172] to show that BEAR converges at a linear rate.

Lemma 13. *Assume Count Sketch has a size $m = \Theta(\varepsilon^{-2}\log 1/\delta)$ with the number of hashes $d = \Theta(\varepsilon^{-1}\log 1/\delta)$. If the instantaneous function $f(\boldsymbol{\beta}, \boldsymbol{\Theta})$ satisfy Assumptions 1 and 2 (Eq.* (6.7) *and* (6.8)*, respectively), then, the corresponding instantaneous function for the sketched problem $f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})$ also satisfy*

$$\frac{p}{m}(1-\varepsilon)M_1\mathbf{I} \preceq \nabla^2_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta}) \preceq \frac{p}{m}M_2(1+\varepsilon)\mathbf{I}, \tag{6.14}$$

$$\mathbb{E}_{\boldsymbol{\theta}}[\|\nabla_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 \mid \boldsymbol{\beta}] \leq \frac{p}{m}M_2(1+\varepsilon)S^2, \tag{6.15}$$

with probability $1-\delta$ each.

Proof. The instantaneous Hessian for the sketched problem (say $\mathbf{H}^s$) is given by

$$\mathbf{H}^s = \nabla^2_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta}) = \mathbf{S}^T\nabla^2 f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\mathbf{S} = \mathbf{S}^T\mathbf{H}\mathbf{S}, \tag{6.16}$$

where $\mathbf{H} = \nabla^2 f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})$ is the instantaneous Hessian for the original problem. Since commuting matrices have the same set of non-zero eigenvalues, the eigenvalues of $\mathbf{S}^T\mathbf{H}\mathbf{S}$ are equal to the eigenvalues of $\mathbf{H}\mathbf{S}\mathbf{S}^T$. Hence,

$$\lambda_{\max}(\mathbf{H}^s) = \lambda_{\max}(\mathbf{S}^T\mathbf{H}\mathbf{S}) = \lambda_{\max}(\mathbf{H}\mathbf{S}\mathbf{S}^T) \tag{6.17}$$

$$\leq \lambda_{\max}(\mathbf{H})\lambda_{\max}(\mathbf{S}\mathbf{S}^T) \tag{6.18}$$

$$\leq M_2\ \lambda_{\max}(\mathbf{S}\mathbf{S}^T) \tag{6.19}$$

$$= M_2\ \lambda_{\max}(\mathbf{S}^T\mathbf{S}), \tag{6.20}$$

where $\lambda_{\max}(\cdot)$ denotes the maximum eigenvalue and $\lambda_{\max}(\mathbf{H}) \leq M_2$ by assumption. Also, Eq. (6.18) uses the fact that the maximum eigenvalue of the product of two symmetric matrices is upper bounded by the product of maximum eigenvalues of individual matrices.

For the count sketch matrix $\mathbf{S} \in \mathbb{R}^{p\times m}$, we have $\mathbb{E}[\mathbf{S}^T\mathbf{S}] = \frac{p}{m}\mathbf{I}$. Moreover, by applying the Matrix-Bernstein inequality [179] on the matrix $\mathbf{Z} = \mathbf{S}^T\mathbf{S} - \frac{p}{m}\mathbf{I} = \sum_{i=1}^{p}\left(\mathbf{S}_i^T\mathbf{S}_i - \frac{1}{m}\mathbf{I}\right) = \sum_{i=1}^{p}\mathbf{Z}_i$, where $\mathbf{S}_i$ is the i-th row in $\mathbf{S}$ and $\mathbf{Z}_i = \left(\mathbf{S}_i^T\mathbf{S}_i - \frac{1}{m}\mathbf{I}\right)\ \forall\ i \in [1, p]$, we get the following bound

$$\mathbb{P}\left(\|\mathbf{Z}\| \geq \epsilon\frac{p}{m}\right) \leq 2m\exp\frac{-\epsilon^2p^2/(2m^2)}{v(\mathbf{Z}) + L\epsilon p/(3m)}$$

where $\|\mathbf{Z}_i\| \leq L\ \forall\ i$ and $v(\mathbf{Z}) = \|\sum_{i=1}^{p}\mathbf{Z}_i^T\mathbf{Z}_i\|$. For the count sketch matrix, we have $L = 1$ and $v(\mathbf{Z}) = \frac{d(k-1)}{k^2}$. Thus, we get

$$\mathbb{P}\left(\|\mathbf{Z}\| \geq \epsilon\frac{p}{m}\right) \leq 2m\exp\frac{-\epsilon^2p^2/(2m^2)}{p(m-1)/m^2 + \epsilon p/(3m)}$$

Further, for any $m = O(\sqrt{p})$, the R.H.S. in the above inequality is upper bounded by δ. Note that the sketch-size m is generally independent and (order-wise) much less than the bigger dimension p, and hence $m = O(\sqrt{p})$ can be easily satisfied by choosing appropriate constants $\epsilon > 0$ and $\delta < 1$. Thus, we get the following bound on the eigenvalues of $\mathbf{S}^T\mathbf{S}$ (also the non-zero eigenvalues in $\mathbf{S}\mathbf{S}^T$)

$$\frac{p}{m}(1-\varepsilon) \leq \lambda_i(\mathbf{S}^T\mathbf{S}) \leq \frac{p}{m}(1+\varepsilon) \text{ for all } i \in [1, m] \tag{6.21}$$

with probability $1-\delta$. Using this in (6.18), we get

$$\lambda_{\max}(\mathbf{H}^s) \leq \frac{p}{m} M_2(1+\varepsilon) \tag{6.22}$$

with probability $1-\delta$.

Similarly, we can write the smallest eigenvalue of $\mathbf{H}^s$ as

$$\lambda_{\min}(\mathbf{H}^s) = \frac{1}{\lambda_{\max}((\mathbf{H}^s)^{-1})} = \frac{1}{\lambda_{\max}((\mathbf{S}^T\mathbf{H}\mathbf{S})^{-1})} = \frac{1}{\lambda_{\max}((\mathbf{H}\mathbf{S}\mathbf{S}^T)^{\dagger})}, \tag{6.23}$$

where $(\cdot)^{\dagger}$ is the Moore-Penrose inverse and the last inequality again uses the fact that commuting matrices have the same set of non-zero eigenvalues. Thus, $\mathbf{S}^T\mathbf{H}\mathbf{S}$ and $\mathbf{H}\mathbf{S}\mathbf{S}^T$, and their corresponding inverses, have the same set of non-zero eigenvalues. Let's define the truncated eigenvalue decomposition of $\mathbf{S}\mathbf{S}^T$ as $\mathbf{S}\mathbf{S}^T = \mathbf{U}\Lambda\mathbf{U}^T$ and note that $(\mathbf{H}\mathbf{S}\mathbf{S}^T)^{\dagger} = (\mathbf{H}\mathbf{U}\Lambda\mathbf{U}^T)^{\dagger} = (\mathbf{U}^T)^{\dagger}\Lambda^{-1}(\mathbf{U})^{\dagger}\mathbf{H}^{-1}$. Hence, we get

$$\begin{aligned}
\lambda_{\max}((\mathbf{H}\mathbf{S}\mathbf{S}^T)^{\dagger}) &= \lambda_{\max}((\mathbf{U}^T)^{\dagger}\Lambda^{-1}(\mathbf{U})^{\dagger}\mathbf{H}^{-1}) \\
&\leq \lambda_{\max}(\Lambda^{-1})\lambda_{\max}((\mathbf{H})^{-1}) \\
&= \lambda_{\max}((\mathbf{S}\mathbf{S}^T)^{\dagger})\lambda_{\max}((\mathbf{H})^{-1})
\end{aligned} \tag{6.24}$$

since Λ^{-1} contains the non-zero eigenvalues of $(\mathbf{S}\mathbf{S}^T)^{\dagger}$. Thus,

$$\begin{aligned}
\lambda_{\min}(\mathbf{H}^s) &\geq \frac{1}{\lambda_{\max}((\mathbf{S}\mathbf{S}^T)^{\dagger})\lambda_{\max}((\mathbf{H})^{-1})} \\
&\geq \lambda_{\min}(\mathbf{S}\mathbf{S}^T)\lambda_{\min}(\mathbf{H}) \\
&\geq \lambda_{\min}(\mathbf{S}\mathbf{S}^T)M_1 \\
&= \lambda_{\min}(\mathbf{S}^T\mathbf{S})M_1 \\
&\geq \frac{p}{m}M_1(1-\varepsilon),
\end{aligned} \tag{6.25}$$

with probability $1-\delta$. where the last inequality follows from (6.21). This proves the desired result.

Similarly, to prove that the gradient of the sketched problem is bounded, observe that $\|\nabla_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 = \|\mathbf{S}^T\nabla f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 \leq \frac{p}{m}M_2(1+\varepsilon)\|\nabla f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2$ with probability $1-\delta$, where the last inequality follows from (6.21). Hence,

$$\mathbb{E}_{\boldsymbol{\theta}}[\|\nabla_{\boldsymbol{\beta}^s} f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 \leq \frac{p}{m}M_2(1+\varepsilon)\mathbb{E}_{\boldsymbol{\theta}}[\|\nabla f(\mathbf{S}\boldsymbol{\beta}^s, \boldsymbol{\Theta})\|^2 \leq \frac{p}{m}M_2(1+\varepsilon)S^2 \tag{6.26}$$

with probability $1-\delta$, where the second inequality follows from assumption in (6.8). □

Finally, to prove Theorem 2, we invoke the results from [172]. According to Theorem 6 in [172], oLBGS with instantaneous functions satisfying assumptions in Eqs. (6.14) and (6.26) converges with probability one. Hence, for BEAR, we get

$$\mathbb{P}(\lim_{t\to\infty} \|\boldsymbol{\beta}_t^s - \boldsymbol{\beta}^{s*}\|^2 = 0) = 1 - \delta. \tag{6.27}$$

Moreover, for the specific step-size $\eta_t = \eta_0/(t + T_0)$, where η_0 and T_0 satisfy the inequality $2m_1\eta_0 T_0 > C$ for some constant C, BEAR satisfies the following rate of convergence (Theorem 7 in [172])

$$\mathbb{E}[f(\boldsymbol{\beta}_t^s, \boldsymbol{\Theta}) - \mathbb{E}[f(\boldsymbol{\beta}^{s*}, \boldsymbol{\Theta})] \leq \frac{C_0}{T_0 + t}, \tag{6.28}$$

with probability $1 - \delta$, where the constant C_0 is given by

$$C_0 = \max\left\{\frac{\eta_0^2 T_0^2 C M_2 p^2 S^2 (1+\varepsilon)^2}{2c^2 m[M_1 p \eta_0 T_0 (1-\varepsilon) - Cm]}, T_0(\mathbb{E}[f(\boldsymbol{\beta}^s, \boldsymbol{\Theta}) - \mathbb{E}[f(\boldsymbol{\beta}^{s*}, \boldsymbol{\Theta})])\right\}.$$

6.9 Discussion and Conclusion

We have developed BEAR, which to the best of our knowledge is the first second-order optimization algorithm for ultra-high dimensional feature selection in sublinear memory. Our results demonstrate that BEAR has up to three orders of magnitude smaller memory footprint compared to the first-order sketching algorithms, which makes it ideal for streaming settings. We showed that the benefits of BEAR is far more pronounced while sketching into lower-dimensional subspaces, which is due to the more accurate decent directions of second-order gradients resulting in less collision-causing noise in Count Sketch. The implications of memory-accuracy advantage of second-order methods goes beyond hashing and streaming and can be applied to improve the communication-computation trade-off in distributed learning in communicating the sketch of the stochastic gradients between nodes [6], [100]. Moreover, while we laid out the algorithmic principles in sketching second-order gradient for training ultra-high dimensional linear classifiers with theoretical guarantees, the same algorithmic principles can be used in training massive-scale nonlinear models such as deep neural networks [67]. We speculate that our work will open up new research directions towards understanding the benefits of second-order optimization in training machine learning models in memory-constrained environments.

Part III

Speeding up Non-Convex Optimization

In this part of the thesis, we study speeding up non-convex optimization for serverless computing:

- **Chapter 7:** Improving training time with data parallelism
- **Chapter 8:** Further improving training time with data and model parallelism

Chapter 7

Stochastic Weight Averaging in Parallel

In this chapter, we propose Stochastic Weight Averaging in Parallel (SWAP), an algorithm to accelerate distributed Deep Neural Network (DNN) training.

7.1 Introduction

Stochastic gradient descent (SGD) and its variants are the de-facto methods to train deep neural networks (DNNs). Each iteration of SGD computes an estimate of the objective's gradient by sampling a *mini-batch* of the available training data and computing the gradient of the loss restricted to the sampled data. A popular strategy to accelerate DNN training is to increase the mini-batch size together with the available computational resources. Larger mini-batches produce more precise gradient estimates; these allow for higher learning rates and achieve larger reductions of the training loss per iteration. In a distributed setting, multiple nodes can compute gradient estimates simultaneously on disjoint subsets of the mini-batch and produce a consensus estimate by averaging all estimates, with one synchronization event per iteration. Training with larger mini-batches requires fewer updates, thus fewer synchronization events, yielding good overall scaling behavior.

Even though the training loss can be reduced more efficiently, there is a maximum batch size after which the resulting model tends to have worse generalization performance [72]–[76]. This phenomenon forces practitioners to use batch sizes below those that achieve the maximum throughput and limits the usefulness of large-batch training strategies.

Stochastic Weight Averaging (SWA) [82] is a method that produces models with good generalization performance by averaging the *weights* of a set of models sampled from the final stages of a training run. As long as the models all lie in a region where the population loss is mostly convex, the average model can behave well, and in practice, it does.

We have observed that if instead of sampling multiple models from a sequence generated by SGD, we generate multiple independent SGD sequences and average models from each, the resulting model achieves similar generalization performance. Furthermore, if all the independent sequences use small-batches, but start from a model trained with large-batches, the resulting model achieves generalization performance comparable with a model trained solely with small-batches.

Using these observations, we derive *Stochastic Weight Averaging in Parallel* (SWAP): A simple strategy to accelerate DNN training by better utilizing available compute resources. Our algorithm is simple to implement, fast and produces good results with minor tuning.

For several image classification tasks on popular computer vision datasets (CIFAR10, CIFAR100, and ImageNet), we show that SWAP achieves generalization performance comparable to models trained with small-batches but does so in time similar to that of a training run with large-batches. We use SWAP on some of the most efficient publicly available models to date, and show that it's able to substantially reduce their training times. Furthermore, we are able to beat the state of the art for CIFAR10 and train in 68% of the time of the winning entry of the DAWNBench competition.[1]

7.2 Related Work

The mechanism by which the training batch size affects the generalization performance is still unknown. A popular explanation is that because of the reduced noise, a model trained using larger mini-batches is more likely to get stuck in a sharper global minima. In [73], the authors argue that sharp minima are sensitive to variations in the data because slight shifts in the location of the minimizer will result in large increases in average loss value. However, if flatness is taken to be the curvature as measured by the second order approximation of the loss, then counterexamples exist. In [180], the authors transform a flat minimizer into a sharp one without changing the behavior of the model, and in [181], the authors show the reverse behavior when weight-decay is not used.

In [72], the authors predict that the batch size can be increased up to a *critical size* without any drop in accuracy and empirically validate this claim. For example, the accuracy begins to drop for image classification on CIFAR10 when the batch sizes exceed 1k samples. They postulate that when the batch size is large, the mini-batch gradient is close to the full gradient, and further increasing the batch size will not significantly improve the signal to noise ratio.

In [74], the authors argue that, for a fixed number of epochs, using a larger batch size implies fewer model updates. They argue that changing the number of updates impacts the distance the weights travel away from their initialization and that this distance determines the generalization performance. They show that by training with large-batches for longer times (thus increasing the number of updates), the generalization performance of the model is recovered. Even though this large-batch strategy generates models that generalize well, it does so in more time than the small-batch alternative.

Irrespective of the generalization performance, the batch size also affects the optimization process. In [182], the authors show that for convex functions in the over-parameterized setting, there is a critical batch size below which an iteration with a batch size of M is roughly equivalent to M iterations with a batch size of one, and batch-sizes larger than M do not improve the rate of convergence.

[1] The https://dawn.cs.stanford.edu/benchmark/

7.3 Stochastic weight averaging in parallel

We describe SWAP as an algorithm in three phases (see Algorithm 16): In the first phase, all workers train a single model by computing large mini-batch updates. Synchronization between workers is required at each iteration and a higher learning rate is used. In the second phase, each worker independently refines its copy of the model to produce a different set of weights. Workers use a smaller batch size, a lower learning rate, and different randomizations of the data. No synchronization between workers is required in this phase. The last phase consists of averaging the weights of the resulting models and computing new batch-normalization statistics to produce the final output.

Phase 1 is terminated before the training loss reaches zero or the training accuracy reaches 100% (for example, a few percentage points below 100%). We believe that stopping early precludes the optimization from getting stuck at a location where the gradients are too small and allows the following stage to improve the generalization performance. However, the optimal stopping accuracy is a hyper-parameter that requires tuning.

During phase 2, the batch size is appropriately reduced and small-batch training is performed independently and simultaneously. Here, each worker (or a subset of them) performs training using all the data, but sampling in different random order. Thus, after the end of the training process, each worker (or subset) will have produced a different model.

Figure 7.1 plots the accuracies and learning-rate schedules for a run of SWAP. During the large-batch phase (phase 1), all workers share a common model and have the same generalization performance. During the small-batch phase (phase 2) the learning rates for all the workers are the same but their testing accuracies differ as the stochasticity causes the models to diverge from each other. We also plot the test-accuracy of the averaged model that would result were we to stop phase 2 at that point. Note that the averaged model performs consistently better than *each individual model*.

7.4 Loss Landscape Visualization around SWAP iterates

To visualize the mechanism behind SWAP, we plot the error achieved by our test network on a plane that contains the outputs of the three different phases of the algorithm. Inspired by [183] and [82], we pick orthogonal vectors u, v that span the plane which contains $\theta_1, \theta_2, \theta_3$. We plot the loss value generated by model $\theta = \theta_1 + \alpha u + \beta v$ at the location (α, β). To plot a loss value, we first generate a weight vector θ, compute the batch-norm statistics for that model (through one pass over the training data), and then evaluate the test and train accuracies.

In Figure 7.2, we plot the training and testing error for the CIFAR10 dataset. Here 'LB' marks the output of phase one, 'SGD' the output of a single worker after phase two, and 'SWAP' the final model. Color codes correspond to error measures at the points interpolated on the plane. In Figure 7.2a, we observe that the level-sets of the training error (restricted to this plane) form an almost convex basin and that both the output of phase 1 ('LB') and the output of one of the workers of phase 2 ('SGD') lie in the outer edges of the basin. Importantly, during phase 2 the model traversed

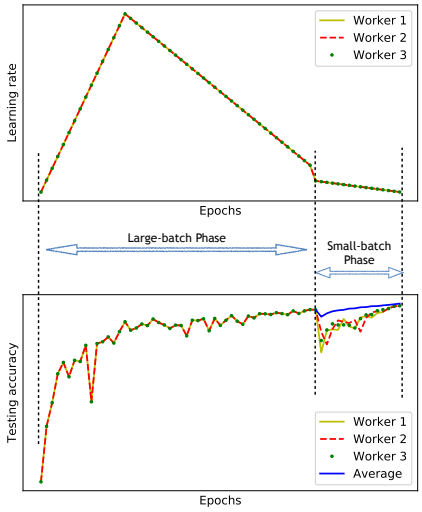


Figure 7.1: Learning rate schedules and CIFAR10 test accuracies for workers participating in SWAP. The large-batch phase with synchronized models is followed by the small-batch phase with diverging independent models. The test accuracy of the averaged weight model is computed by averaging the independent models and computing the test loss for the resulting model.

to a *different side* of the basin (and not to the center). Also, the final model ('SWAP') is closer to the center of the basin.

When we visualize these three points on the test loss landscape (Figure 7.2b), we observe that the variations in the topology of the basin cause the 'LB' and 'SGD' points to fall in regions of higher error. But, since the 'SWAP' point is closer to the center of the basin, it is less affected by the change in topology. In Figure 7.3, we neglect the 'LB' point and plot the plane spanned by three workers 'SGD1', 'SGD2', 'SGD3'. In Figure 7.3a, we can observe that these points lie at different sides of the training error basin while 'SWAP' is closer to the center. In Figure 7.3b, we observe that the change in topology causes the worker points to lie in regions of higher testing errors than 'SWAP', which is again close to the center of both basins. For reference, we have also plotted the best model that can be generated by this region of the plane.

7.4.1 Sampling from independent runs of SGD or sampling from one

In [184], the authors argue that in the later stages of SGD the weight iterates behave similar to an Ornstein Uhlenbeck process. So, by maintaining a constant learning rate the SGD iterates should reach a stationary distribution that is similar to a high-dimensional Gaussian. This distribution is centered at the local minimum, has a covariance that grows proportionally with the learning rate, inversely proportional to the batch size and has a shape that depends on both the Hessian of the mean loss and covariance of the gradient.

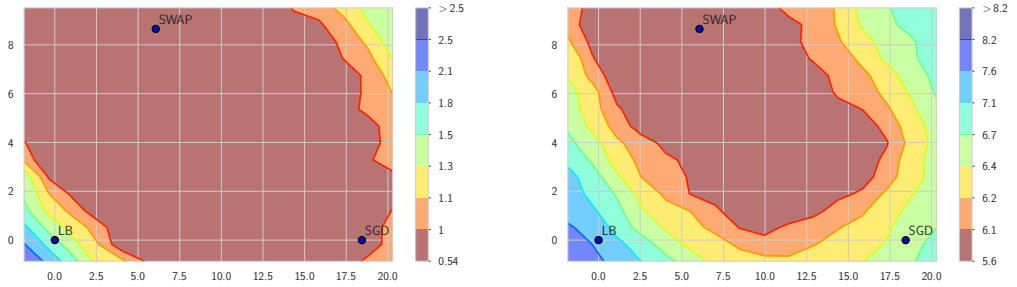
The authors of [82] argue that by virtue of being a high dimensional Gaussian all the mass of the distribution is concentrated near the 'shell' of the ellipsoid, and therefore, it is unlikely for SGD to access the interior. They further argue that by sampling weights from an SGD run (leaving enough

Algorithm 16: Stochastic Weight Averaging in Parallel (SWAP)

Number of workers W; Weight initialization θ_0; $t = 0$
Training accuracy, τ, at which to exit phase one
Learning rate schedules LR_1 and LR_2 for phase one and two, respectively
Mini-batch sizes B_1 and B_2 for phase one and two, respectively
Gradient of loss function for sample i at weight θ: g^i
$\text{Update}(\cdot)$: A function that computes a weight update from a history of gradients
Phase 1:
while *Training accuracy* $\leq \tau$ **do**
    $\eta \leftarrow LR_1(t)$
    for w *in* $[0, ..., W-1]$ *In parallel* **do**
        $B^w \leftarrow$ random sub-sample of training data with size $\frac{B_1}{W}$
        $g^w \leftarrow \frac{W}{|B_1|} \sum_{i \in B^w} g^i$ worker gradient
    end
    $g_t \leftarrow \frac{1}{W} \sum g^w$ synchronization of worker gradients
    $\theta_{t+1} = \theta_t + \text{Update}(\eta_t, g_t, g_{t-1}, \cdots)$; `/* first order method update */`
    $t = t + 1$; $T = t$
end
Phase 2:
for t *in* $[T, T + Q]$ **do**
    $\eta \leftarrow LR_2(t - T)$
    for w *in* $[0, ..., W-1]$ *In parallel* **do**
        $B^w \leftarrow$ random sub-sample of training data with size B_2
        $g^w \leftarrow \frac{1}{|B_2|} \sum_{i \in B^w} g^i$ worker gradient
        $\theta^w_{t+1} = \theta^w_t + \text{Update}(\eta_t, g^w_t, g^w_{t-1}, \cdots)$; `/* first order method update at local worker */`
    end
end
`/* We get` W `different models at the end of phase 2 */`
Phase 3: $\hat{\theta}_\ell \leftarrow \frac{1}{W} \sum \theta^i_{T+Q}$ produce averaged model
Compute batch-norm statistics for $\hat{\theta}_\ell$ to produce θ_ℓ
Result: Final model θ_ℓ

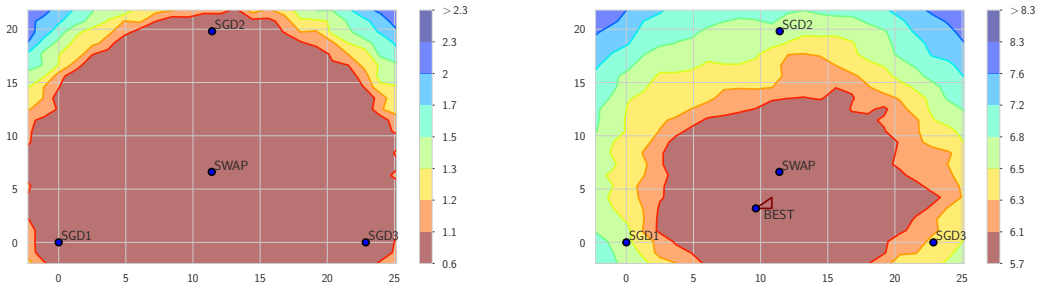
time steps between them) will choose weights that are spread out on the surface of this ellipsoid and their average will be closer to the center.

Without any further assumptions, we can justify sampling from different SGD runs (as done in phase 2 during SWAP). As long as all runs start in the same basin of attraction, and provided the model from [184] holds, all runs will converge to the same stationary distribution, and each run can generate independent samples from it.



(a) Train Error (%) (b) Test Error (%)

Figure 7.2: CIFAR10 train and test error restricted to a 2D plane spanned by the output of phase 1 ('LB') one of the outputs of phase 2 ('SGD') and the averaged model ('SWAP').



(a) Train Error (%) (b) Test Error (%)

Figure 7.3: CIFAR10 train and test error restricted to a 2D plane spanned by the output of three workers after phase 2 ('SGD1', 'SGD2', 'SGD3') and location of the average model ('SWAP'). The minimum test error achievable for models restricted to this region of the plane (marked as *BEST*).

7.4.2 Orthogonality of the gradient and the direction to the center of basin

To win some intuition on the advantage that SWA and SWAP have over SGD, we measure the cosine similarity between the gradient descent direction, $-g_i$, and the direction towards the output of SWAP, $\Delta\theta = \theta_{\text{swap}} - \theta_i$. In Figure 7.4, we see that the cosine similarity, $\frac{\langle \Delta\theta, -g_i \rangle}{\|g_i\| \|\Delta\theta\|}$, decreases as the training enters its later stages. We believe that towards the end of training, the angle between the gradient direction and the directions toward the center of the basin is large, therefore the process moves mostly orthogonally to the basin, and progress slows. However, averaging samples from different sides of the basin can (and does) make faster progress towards the center.

7.5 Experiments

In this section we evaluate the performance of SWAP for image classification tasks on the CIFAR10, CIFAR100, and ImageNet datasets.

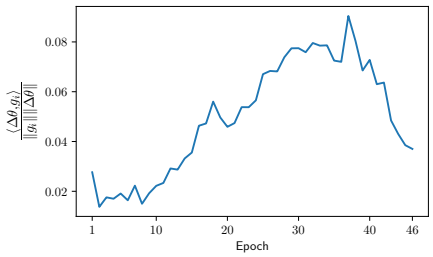


Figure 7.4: Cosine similarity between direction of gradient descent and $\Delta\theta$

7.5.1 CIFAR10 and CIFAR100

For the experiments in this subsection, we found the best hyper-parameters using grid searches (see Tables 7.5 and 7.6 for details). We train using mini-batch SGD with Nesterov momentum (set to 0.9) and weight decay of 5×10^{-4}. We augment the data using cutout [185] and use a fast-to-train custom ResNet 9 from a submission [2] to the DAWNBench leaderboard [186]. All experiments were run on one machine with 8 NVIDIA Tesla V100 GPUs and use Horovod [187] to distribute the computation. All statistics were collected over 10 different runs.

CIFAR10: For these experiments, we used the following settings—SWAP phase one: 4096 samples per batch using 8 GPUs (512 samples per GPU). Phase one is terminated when the training accuracy reaches 98% (on average 108 epochs). SWAP phase two: 8 workers with one GPU each and 512 samples per batch for 30 epochs. The experiment that uses only large-batches had 4096 samples per batch across 8 GPUs and is run for 150 epochs. The experiments that use only small-batches had 512 samples per batch on 2 GPUs and is trained for 100 epochs.

Table 7.1 compares the best test accuracies and corresponding training times for models trained with small-batch only, with large-batch only, and with SWAP. We report the average accuracy of the workers before averaging and the accuracy of the final model.

CIFAR10	**Test Accuracy (%)**	**Training Time (sec)**
SGD (small-batch)	95.24 ± 0.09	254.12 ± 0.62
SGD (large-batch)	94.77 ± 0.23	132.62 ± 1.09
SWAP (before averaging)	94.70 ± 0.20	167.57 ± 3.25
SWAP (after averaging)	95.23 ± 0.08	169.20 ± 3.25

Table 7.1: Training Statistics for CIFAR10

CIFAR100: For these experiments, we use the following settings—SWAP phase one: 2048 samples per batch using 8 GPUs (256 samples per GPU). Phase one exits when the training accuracy reaches 90% (on average 112 epochs). SWAP phase two: 8 workers with one GPU each and 128 samples per batch, training for for 10 epochs. The experiments that use only large-batch training were run for 150 epochs with batches of 2048 on 8 GPUs The experiments that use only small-batch were trained for 150 epochs using batches of 128 on 1 GPU.

[2] https://github.com/davidcpage/cifar10-fast

CIFAR100	Test Accuracy (%)	Training Time (sec)
SGD (small-batch)	77.01 ± 0.25	573.76 ± 2.25
SGD (large-batch)	75.84 ± 0.35	116.13 ± 1.35
SWAP (before averaging)	75.74 ± 0.15	123.11 ± 1.85
SWAP (after averaging)	78.18 ± 0.21	125.34 ± 1.85

Table 7.2: Training Statistics for CIFAR100

Table 7.2 compares the best test accuracies and corresponding training times for models trained with only small-batches (for 150 epochs), with only large-batches (for 150 epochs), and with SWAP. For SWAP, we report test accuracies obtained using the last SGD iterate before averaging, and test accuracy of the final model obtained after averaging. We observe significant improvement in test accuracies after averaging the models.

For both CIFAR 10 and CIFAR100, training with small-batches achieves higher testing accuracy than training with large-batches but takes much longer to train. SWAP, however, terminates in time comparable to the large-batch run but achieves accuracies on par (or better) than small batch training.

Achieving state of the art training speeds for CIFAR10: At the time of writing the front-runner of the DAWNBench competition takes 37 seconds with 4 Tesla V100 GPUs to train CIFAR10 to 94% test accuracy. Using SWAP with 8 Tesla V100 GPUs, a phase one batch size of 2048 samples and 28 epochs, and a phase two batch size of 256 samples for one epoch is able to reach the same accuracy in 27 seconds.

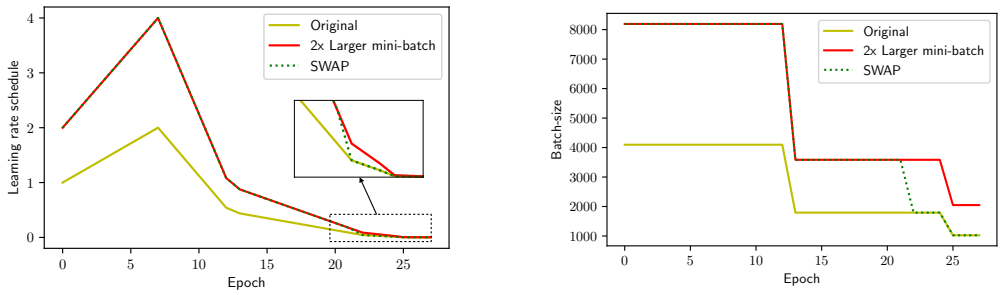
7.5.2 Experiments on ImageNet

We use SWAP to accelerate a publicly available fast-to-train ImageNet model with published learning rate and batch size schedules [3]. The default settings for this code modify the learning-rates and batch sizes throughout the optimization (see Figure 7.5). Our small-batch experiments train ImageNet for 28 epochs using the published schedules with no modification and are run on 8 Tesla V100 GPUs. Our large-batch experiments modify the schedules by doubling the batch size and doubling the learning rates (see Figure 7.5) and are run on 16 Tesla V100 GPUs. For SWAP phase 1, we use the large-batch settings for 22 epochs, and for SWAP phase 2, we run two independent workers each with 8 GPUs using the settings for small-batches for 6 epochs.

We observe that doubling the batch size reduces the Top1 and Top5 test accuracies with respect to the small-batch run. SWAP, however, recovers the generalization performance at substantially reduced training times. Our results are compiled in Table 7.3 (the statistics were collected over 3 runs). We believe it's worthy of mention that these accelerations were achieved with no tuning other than increasing the learning rates proportionally to the increase in batch size and reverting to the original schedule when transitioning between phases. Note that there exist training schemes in the

[3] Available at https://github.com/cybertronai/imagenet18_old

literature that train on even larger batch sizes such as 32k [79], [81], but these methods require a lot of hyperparameter tuning specific to the dataset.



(a) Learning rate schedule

(b) Batch sizes across epochs for ImageNet

Figure 7.5: Learning rate and mini-batch schedules used for ImageNet. The original schedule for 8 GPUs was taken from an existing DAWNBench submission. For a larger batch experiment, we double the batch size, double the number of GPUs and double the learning rate of the original schedule. For SWAP, we switch from the modified schedule to the original schedule as we move from phase 1 to phase 2.

ImageNet	Top1 Accuracy (%)	Top5 Accuracy (%)	Training Time (min)
SGD (small-batch)	76.14 ± 0.07	93.30 ± 0.07	235.29 ± 0.33
SGD (large-batch)	75.86 ± 0.03	92.98 ± 0.06	127.20 ± 0.78
SWAP (before averaging)	75.96 ± 0.02	93.15 ± 0.02	149.12 ± 0.55
SWAP (after averaging)	76.19 ± 0.03	93.32 ± 0.02	156.55 ± 0.56

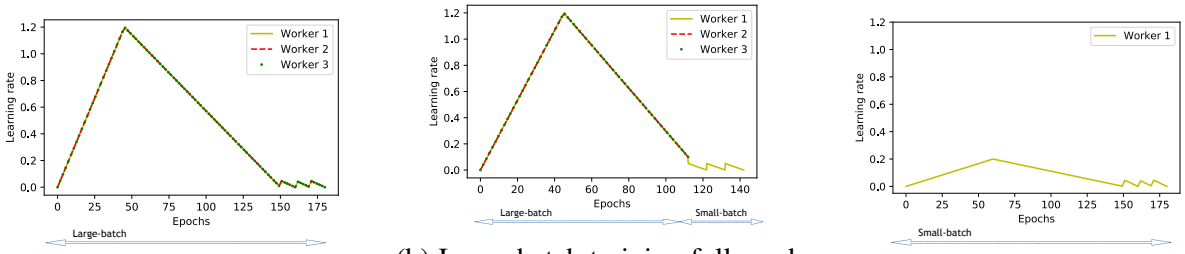
Table 7.3: Training Statistics for ImageNet

7.5.3 Empirical comparison of SWA and SWAP

We now compare SWAP with SWA: the sequential weight averaging algorithm from [82]. For the experiments in this section, we use the CIFAR100 dataset. We sample the same number of models for both SWA and SWAP and maintain the same number of epochs per sample. For SWA, we sample each model with 10 epochs in-between and average them to get the final model. For SWAP, we run 8 independent workers for 10 epochs each and use their average as the final model.

Large-batch SWA: We explore if SWA can recover the test accuracy of small-batch training on a large-batch training run. We use the same (large) batch size throughout. We follow an initial training cycle with cyclic learning rates (with cycles of 10 epochs) to sample 8 models (one from the end of each cycle). See Figure 7.6a for an illustration of the learning rate schedule.

As expected we observe that the large-batch training run achieves lower training accuracy, but surprisingly SWA was unable to improve it (see Table 7.4, row 1).



(a) Large-batch SWA

(b) Large-batch training followed by SWA with small-batches

(c) Small-batch SWA

Figure 7.6: Illustration of SWA with different batch sizes

Large-batch followed by small-batch SWA: We evaluate the effect of executing SWA using small-batches after a large-batch training run. We interrupt the large-batch phase at the same accuracy we interrupt phase 1 of our CIFAR100 experiment (Table 7.2). In this case, the small-batch phase uses a single worker and samples the models sequentially. SWA is able to reach the test accuracy of a small-batch run but requires more than three times longer than SWAP to compute the model (see Table 7.4, row 2). An illustration of the learning rate schedule is provided in Figure 7.6b.

Small-batch SWA and SWAP: We start the SWA cyclic learning rate schedule from the best model found by solely small-batch training (table 7.2, row 1). Since the cycle length and cycle count are fixed, the only free parameter is the peak learning rate. We select this using a grid-search. Once the SWA schedule is specified, we re-use the peak learning rate settings in SWAP. We start phase two from the model that was generated as the output of phase 1 for the experiment on section 7.5.1 reported on table 7.2 rows 3 and 4. With these settings, small-batch SWA achieves better accuracy than SWAP (by around $\sim 0.9\%$) at 6.8x more training time.

Next, we wish to explore the speed-up that SWAP achieves over SWA if the precision of SWA is set as a target. To that end, we relax the constraints on SWAP. By increasing the phase two schedule from one 10 epoch cycle to two 20 epoch cycles and sampling two models from each worker (16 models) the resulting model achieved a test accuracy of 79.11% in 241 seconds or 3.5x less time.

CIFAR100	Test accuracy before averaging (%)	Test accuracy after averaging (%)	Training Time (sec)
Large-batch SWA	76.06 ± 0.25	76.00 ± 0.31	376.4 ± 2.25
Large-batch followed by small-batch SWA	76.26 ± 0.35	78.12 ± 0.14	398.0 ± 1.35
Small-batch SWA	76.80 ± 0.15	79.09 ± 0.19	848.6 ± 5.61
SWAP (10 small-batch epochs)	75.74 ± 0.15	78.18 ± 0.21	125.30 ± 1.85
SWAP (40 small-batch epochs)	76.19 ± 0.19	79.11 ± 0.12	241.54 ± 1.62

Table 7.4: Comparison: SWA versus SWAP

Hyperparameters for CIFAR10 and CIFAR100 Experiments: We provide the parameters used in the experiments of Section 7.5.1. These were obtained by doing independent grid searches

for each experiment. For all CIFAR experiments, the momentum and weight decay constants were kept at 0.9 and 5×10^{-4} respectively. Tables 7.5 and 7.6 list the remaining hyperparameters. When a stopping accuracy of 100% is listed, we mean that the maximum number of epochs were used.

CIFAR10	**SGD (small-batch)**	**SGD (large-batch)**	**SWAP (Phase 1)**	**SWAP (Phase 2)**
Batch-size	512	4096	4096	512
Learning-rate Peak	0.3	1.2	1.2	0.12
Maximum Epochs	100	150	150	30
Warm-up Epochs	30	30	30	0
GPUs used per model	2	8	8	1
Stopping Accuracy (%)	100	100	98	100

Table 7.5: Hyperparameters obtained using tuning for CIFAR10

CIFAR100	**SGD (small-batch)**	**SGD (large-batch)**	**SWAP (Phase 1)**	**SWAP (Phase 2)**
Batch-size	128	2048	2048	128
Learning-rate Peak	0.2	1.2	1.2	0.05
Total Epochs	150	150	150	30
Warm-up Epochs	60	45	45	0
GPUs used per model	1	8	8	1
Stopping Accuracy (%)	100	100	90	100

Table 7.6: Hyperparameters obtained using tuning for CIFAR100

7.6 Conclusions and Future Work

We propose Stochastic Weight Averaging in Parallel (SWAP), an algorithm that uses a variant of Stochastic Weight Averaging (SWA) to improve the generalization performance of a model trained with large mini-batches. Our algorithm uses large mini-batches to compute an approximate solution quickly and then refines it by averaging the weights of multiple models trained using small-batches. The final model obtained after averaging has good generalization performance and is trained in a shorter time. We believe that this variant and this application of SWA are novel.

We observed that using large-batches in the initial stages of training does not preclude the models from achieving good generalization performance. That is, by refining the output of a large-batch run, with models sampled sequentially as in SWA or in parallel as in SWAP, the resulting model is able to perform as well as the models trained using small-batches only. We confirm this in the image classification datasets CIFAR10, CIFAR100, and ImageNet.

Through visualizations, we complement the existing evidence that averaged weights are closer to the center of a training loss basin than the models produced by stochastic gradient descent. It's

interesting to note that the basin into which the large mini-batch run is converging to seems to be the same basin where the refined models are found. So, it is possible that regions with bad and good generalization performance are connected through regions of low training loss and, more so, that both belong to an almost convex basin. Our method requires the choice of (at least) one more hyperparameter: the transition point between the large-batch and small-batch. For our experiments, we chose this by using a grid search. A principled method to choose the transition point will be the focus of future work.

Chapter 8

Dynamic Communication Thresholding

In this chapter, we consider hybrid parallelism—a paradigm that employs both Data Parallelism (DP) and Model Parallelism (MP) to scale distributed training of deep neural networks.

8.1 Introduction

Data Parallelism (DP), in which each (of many) trainers stores a replica of the entire model, is a popular parallelization paradigm for the training of very large Deep Neural Networks (DNNs) [188], [189]. At the beginning of each training iteration, each worker processes a subset of entire training data with a predetermined batch size, and then each worker synchronizes the model parameters at the end of the iteration. DP has experienced widespread deployment, but it is now facing two major challenges. The first challenge is that large batch size is needed to exploit fully the ever-increasing compute power of training nodes. This turns out to be difficult. Both theoretical and empirical evidence suggests that going beyond a certain batch size for training DNNs results in loss in generalization performance [72]–[76], [182], [190], [191]. Despite active research on restoring generalization performance when the batch size is large [60], [61], [77]–[81], [192], these methods either are specific to certain models and/or datasets, require extensive hyperparameter tuning, or can at best increase the maximum batch size by a small factor. The second challenge is that due to increasing model complexity and parameters in domains such as, but not limited to, natural language processing and recommendation systems (e.g., see [87], [193]–[195]), coupled with the saturation of single machine memory and compute power due to trends such as the ending of Moore's law [196], [197], replication of an entire DNN model on each worker becomes an increasingly infeasible proposition.

For these reasons, Model Parallelism (MP) as an alternative parallelization paradigm has gained significant traction both from the industry and the research community in the last several years [198]–[203]. In its purest form, the entire network during MP is partitioned into a number of sub-networks equal to the number of workers. While this form can accommodate a larger network than DP, it fails to capitalize on the largest batch size that is allowable before generalization performance degrades.

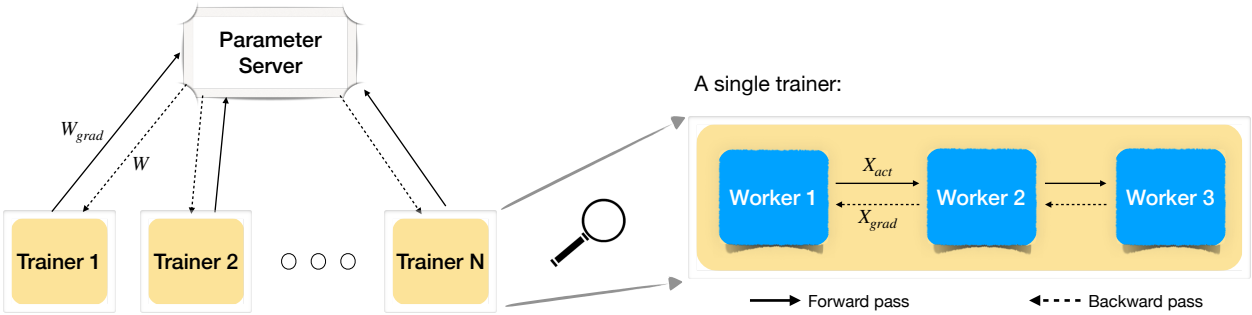


Figure 8.1: Distributed DNN training with hybrid training which uses both DP (left) and MP (right) for greater parallelization gains. During DP, multiple trainers process several mini-batches of data in parallel. During MP, one copy of the model is processed by one trainer which in turn is comprised of multiple workers.

Hybrid Parallelism (HP)—that employs both DP and MP—is a natural next step, an idea that was arguably first introduced in [188], and more recently exploited further for large-scale DNN training [204]–[209]. An illustration of hybrid training that uses MP to distribute the model across workers and DP to process multiple batches of training data at once is provided in Fig. 8.1. Here, each partition of the network for MP is replicated in a group of workers, each processing the entire batch for that sub-network in question. The scaling of model size and batch size by HP has now progressed to the next bottleneck: Communication bandwidth [198]. We are running into this communication bottleneck issue that has hitherto not been discussed in the works referenced above. The bottleneck exists in two crucial places. First, for MP, activation values and gradient information need to be communicated from one sub-network to the next during forward and backward propagation. Second, for DP, gradients of the same sub-network but for different sub-batches need to be communicated, regardless of the exact operations that follow. This depends on the specific communication protocol (centralized versus decentralized reduction) or the algorithm (synchronous versus asynchronous updates). To compound the problem further, increasing the batch size to fully exploit DP increases the communication of activations and gradients in MP, the sizes of which are directly proportional to the batch size. Additionally, in the asynchronous training, increasing batch size exacerbates the stale gradient problem due to an increase in the time interval between a worker receiving the model and sending the gradient [210]. In short, the benefits of communication reduction are many.

Dynamic Communication Thresholding. We propose a Dynamic Communication Thresholding (DCT) framework for communication-efficient training for HP. It incorporates two algorithms, DCT-DP and DCT-MP, to alleviate communication congestion for DP and MP, respectively. Our algorithms filter the entities to be communicated through a simple hard-thresholding function, eliminating the need to pass many of them over the communication fabric. We propose practical methods to compute the thresholds to reduce the computational overhead of compression. Our thresholding technique is versatile, as it applies to different communication primitives in DP for the gradients, to different pipelining methods in MP (e.g., GPipe [199], PipeDream [198]), and to different applications such as recommendation systems and natural language processing models. While

thresholding communication introduces errors, we apply (previously known) error compensation technique as well as a model consistency adjustment method (we developed) to mitigate the effect of the error in compression. Consequently, despite significant communication thresholding, model accuracy does not degrade, and in fact it often improves. Overall, communication costs are reduced by factors of up to $20\times$, and end-to-end training time is cut by as much as 37% for large-scale model training.

Related Work. Due to the use of large clusters with powerful machines to train complex DNNs (e.g. BERT-Large [87] with 340M parameters), the distributed training workloads are becoming increasingly communication bound. Thus, numerous compression schemes have been proposed in the past several years for the data parallel setting (see [88] for a detailed survey).

When it comes to performance on real-world systems, many of these existing schemes have one or more of the following shortcomings. (i) Focus is mostly on a theoretical analysis of schemes based on restricted assumptions, such as convexity and synchronous SGD. (ii) The empirical evaluation ignores the cost of compression and decompression which, in many cases, deprives them of any savings due to communication. (iii) Comparison of convergence with respect to baseline is reported, while the number of epochs (or iterations) and the actual training time is ignored. For instance, in Fig. 1 in [88], the authors compare the compression scheme in [99] with a baseline without compression. They observe that, although the convergence with respect to the number of epochs is unaffected due to compression, it takes almost twice the time for training to converge, rendering the scheme worse than no compression. We also observed in our experiments that for sparsification using top-K sparsity [93], [94], the overhead of copying and sorting the large vectors ends up taking more time than the gains obtained due to communication reduction. (See Sec. 8.3 for details.) In this paper, we propose practical schemes for communication reduction, and we show performance improvements in terms of the end-to-end DNN training times, with loss that is similar to, or in some cases better than, the baseline.

For the MP case, existing works target the scheduling of communication of entities across the network to improve the efficiency of training DNNs [211], [212]. However, to the best of our knowledge, this is the first work that targets communication reduction for MP by explicitly compressing the entities that are sent across the network. As such, it can be applied on top of existing training efficiency schemes, such as communication scheduling [211], [212] and Pipelining [198], [199], [202], [213] for MP. As illustrated in Fig. 8.1 (right), communication is a major bottleneck for MP-based training since the activations are communicated from (say) worker 1 to worker 2 during the forward pass and the gradients are then communicated from worker 2 to worker 1 during the backward pass (similar communication happens between workers 2 and 3). However, we further observed that naively applying compression schemes, such as sparsification, quantization and sketching, to the activations and gradients either do not achieve high enough compression rates to be practical, or the degradation in model performance is beyond an acceptable level. (See Appendix 8.4 for details on such negative results.)

In the next section, we describe our algorithms for communication efficiency during parallelization, for both the MP and DP primitives of the DNN training. In particular, we discuss DCT-DP (in Section 8.2.1) and explain our gradient reduction technique for DP that requires minimal computational overhead for compression; and then we discuss DCT-MP (in Section 8.2.2), a flexible

thresholding framework with theoretical support for our design. Then, Section 8.3 reports our findings from a diverse set of experiments and demonstrates the advantages of using DCT-DP and DCT-MP for training large-scale models.

8.2 Communication-Efficient Training with Hybrid Parallelism

We start, in Section 8.2.1, by proposing a Dynamic Communication Thresholding (DCT) technique for DP (DCT-DP). DCT-DP is inspired by existing theoretical works such as [93] and [94]. It sparsifies the gradient in each iteration before sending it over the wire, and it intelligently chooses the threshold for sparsification to reduce the computational overhead introduced due to compression and decompression. Then, in Section 8.2.2, we propose DCT-MP, a novel thresholding scheme for sparsification of activations and gradients during forward and backward passes, respectively, to reduce communication during MP.

8.2.1 DCT-DP: Reducing communication for Data Parallelism

During DP, as illustrated in Fig. 8.1 (left), we compress the gradient, W_{grad}, from trainers to the parameter server to improve the communication bottleneck. Our compression algorithm, DCT-DP, is inspired by previous works which focus on data-parallel training for alleviating communication bottlenecks, and in particular the works of [93], [94], where error feedback is employed along with sparsification to correct the error in gradient direction due to compression. Such schemes find a top-K threshold by sorting the gradient vector, and they use the threshold to sparsify the gradients by keeping only the top-K entries. However, they focus on proving theoretical convergence guarantees, and they do not show improvements in end-to-end times for training neural networks.

In our experiments, we observed that the overhead of allocating memory to copy the gradient (with its size easily scaling into the millions) and sorting the resultant copy to find the top-K threshold in each iteration is sufficiently expensive that it deprives any improvements in end-to-end training time in real-world systems (see Sec. 8.3.3 for details). Hence, such gradient compression schemes, in their most basic form, cannot be employed directly to obtain promised gains in training efficiency. However, we take advantage of the following observation to reduce the overhead introduced due to compression.

In Fig. 8.2, we plot the top-K thresholds for various levels of sparsity for the Deep Learning Recommendation Model (DLRM) [195] with the Criteo Ad Kaggle Dataset for one of the Fully Connected (FC) layers (see Sec. 8.3.1 for details on the training process). We see that the threshold value increases as the sparsity increases, which is expected. More importantly, we note that given a sparsity factor, the threshold value does not vary much across iterations. For example, for 95% sparsity, the threshold deviates by at most 26% around its running mean. Thus, even for reasonably large compression factors, updating the threshold every iteration is excessive.

Inspired by this observation, we update the threshold only once every L iterations (where L is generally in thousands) while compressing the gradient of the parameters, W_{grad}, for each DNN layer. We refer to L as the threshold life-span. As we observe in our experiments (see Sec. 8.3.3), we

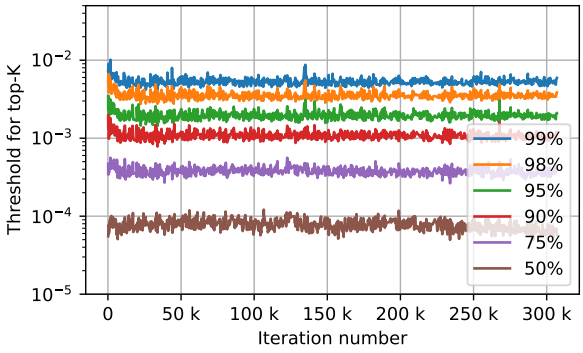


Figure 8.2: Top-K threshold for various levels of sparsity during the gradient compression for DCT-DP. We see that the top-K thresholds, for different sparsity levels, do not deviate much from the mean. Thus, updating the threshold only every $L(> 1)$ iterations can help reduce the overhead of sorting to find the top-K threshold.

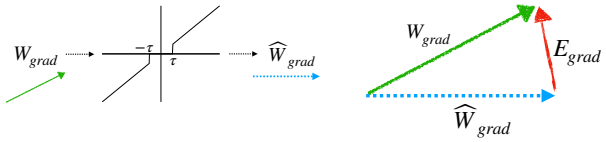


Figure 8.3: A illustration of DCT-DP. First, $W_{grad} \in \mathbb{R}^N$ (which already incorporates error from the previous iteration) is compressed using a threshold τ to obtain the sparse vector $\widehat{W}_{grad}$. Then, the error is calculated as $E_{grad} = W_{grad} - \widehat{W}_{grad}$ to be used in the next iteration to correct the error in gradient direction.

can compress the gradients by as much as 99% sparsity with $L = 1000$ for each layer using top-K sparsification and error correction without any loss in performance. Our algorithm is illustrated in Fig. 8.3 and detailed steps are provided in Algorithm 11. Throughout this paper, the function $\mathbb{I}(\cdot)$ denotes the indicator function, and the symbols $\lfloor \cdot \rfloor$ and $\odot$ denote the integer floor and element-wise product of two matrices, respectively.

Note that each trainer consists of multiple workers, and each worker compresses the gradients layer-wise using sparsification before communication (see Fig. 8.1 for an illustration, where each trainer consists of 3 workers). This is unlike existing works (e.g. [93], [100]) where the gradient vectors of all the model parameters are combined and compressed together. However, the theoretical guarantees on the convergence of the algorithm still holds and can be trivially extended to our case. This is because, for any threshold $\tau > 0$, the compressed gradient satisfies the contraction property (Definition 2.1 in [93]). Hence, DCT-DP satisfies the same rate of convergence as Stochastic Gradient Descent (SGD) without compression (see Theorem 2.4 in [93]).

Algorithm 17: DCT-DP: Communication-Efficient Data Parallelism

Input: Sparsity factor η ($0 < \eta \leq 1$), Threshold life-span L, Iteration number k, Gradient of the DNN layer $W_{grad} \in \mathbb{R}^N$, Error $E_{grad} \in \mathbb{R}^N$, and Threshold τ (from iteration $k-1$)
Error Feedback: $W_{grad} = W_{grad} + E_{grad}$
if *L divides k* **then**
    $[w_1, w_2, \cdots, w_N]$ = $Sort(|W_{grad}|)$
    Assign $\tau = w_{\lfloor N \times \eta \rfloor}$
else
    Use τ from iteration $k-1$
Compute mask $M = \mathbb{I}(|W_{grad}| \geq \tau)$
Compute compressed gradient $\widehat{W}_{grad} = W_{grad} \odot M$
Compute error $E_{grad} = W_{grad} - \widehat{W}_{grad}$
Send $\widehat{W}_{grad}$ to the parameter server which updates the model

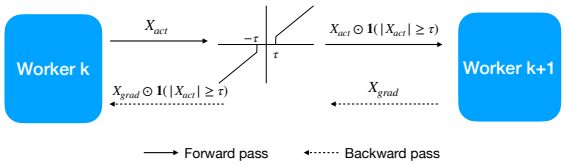


Figure 8.4: A illustration of DCT-MP. During the forward pass, we sparsify and compress the activations, say X_{act}, corresponding to one data sample, using the mask, $\mathbb{I}(|X_{act}| \geq \tau)$, is generated based on the threshold τ. During the backward pass, the same mask is used to compress the gradients and selectively train neurons.

8.2.2 DCT-MP: Reducing communication for Model Parallelism

Training of large-scale DNNs is often regarded with pessimism due to its associated training latency (multiple days/weeks). However, training such large-scale models can be a "blessing in disguise" from a communication-efficiency point of view. For such models, with billions of parameters in each layer, only a few of the neurons are activated during the forward pass, potentially allowing us to compress these activations by a factor of $20\times$ or more with no loss in model performance. This idea of training only a subset of neurons every iteration based their activation values stems from several existing observations [214]–[216]. In fact, in works such as dropout [217] and adaptive dropout [218], the authors have shown that selective sparsification can improve the generalization performance due to implicit regularization [219]. With such a scheme, we also observe gains in generalization performance on top of communication efficiency (see experiments in Section 8.3).

Motivated by this, we propose a sparsification scheme where the neurons compete with each other in every iteration during DNN training, and the ones with the largest (absolute) value of activations are selected. Thus, for a given training sample, DCT-MP selects only a few neurons (say $\sim$5%) during the forward pass that are generally sufficient to represent the entire information for

Algorithm 18: DCT-MP: Communication-Efficient Model Parallelism

Input: Sparsity factor η ($0 < \eta \leq 1$),
Forward Pass:
Input: Activation matrix $X_{act} = [X_{act,i}]_{i=1}^{B} \in \mathbb{R}^{B \times d}$
Define the mask, $M = [\,]$
for $i = 1$ ***to*** B **do**
  $[x_1, x_2, \cdots, x_d]$ = $Sort(|X_{act,i}|)$
  Define $\tau_i = x_{\lfloor d \times \eta \rfloor}$
  $m_i = \mathbb{I}(|X_{act,i}| \geq \tau_i)$
  $M = [M;\ m_i]$
Compute the sparse matrix $X_{act} \odot M$
Send $X_{act} \odot M$ and the mask M across the network
Backward Pass:
Input: Gradient matrix $X_{grad} \in \mathbb{R}^{B \times d}$
Compute the sparse matrix $X_{grad} \odot M$
Send $X_{grad} \odot M$ across the network

that training sample. We next describe DCT-MP in more detail.

Algorithm. Let the mini-batch size be B and the number of output features before the model split be d. Thus, the activation and gradient matrices (X_{act} and X_{grad}, respectively) lie in $\mathbb{R}^{B \times d}$. Based on the idea that each example activates only a subset of neurons, we select a fraction, say η, of largest entries according to their absolute value in each row. Thus, for the i-th row of X_{act}, say $X_{act,i}$, we select a threshold τ_i which is greater than $d \times \eta$ values in $X_{act,i}$, and the mask is thus calculated for the i-th data sample as $\mathbb{I}(X_{act,i} \geq \tau_i)$. The same mask is then used to compress the entities $X_{act,i}$ and $X_{grad,i}$ during forward and backward passes, respectively, for all $i \in \{1, 2, \cdots, B\}$. Thus, the training for each mini-batch happens only on the relevant neurons corresponding to each sample in the training data. In Fig. 8.4, we illustrate the compression using DCT-MP when the mini-batch size is one. Detailed steps for a general mini-batch size B are provided in Algorithm 18.

DCT-MP Promotes Sparsity in Model Activations. In Fig. 8.5, we plot the mean, $\frac{1}{B}\sum_{i=1}^{B} \tau_i$, of threshold vector $\tau = [\tau_1, \tau_2, \cdots, \tau_B]$ with respect to the number of iterations for the DLRM model with the Criteo Ad Kaggle Dataset. The threshold is calculated for activations after one of the fully connected layers (see Sec. 8.3.1 for details on the experimental setup). The mean of the threshold is calculated for different sparsity levels ($75\%, 90\%$ and 95%) for the two cases when sparsification using DCT-MP is applied (dotted lines) and when it is not applied (solid lines). Thus, the solid lines correspond to a single training run where we are simply measuring the mean of top-K threshold values without actually sparsifying the activations sent across the wire. The dotted lines with different sparsification levels correspond to different training runs where the stated sparsification is actually applied to the activations (and gradients) that are sent across the wire.

We observe that, as the training progresses, the top-K thresholds decrease significantly faster for the case when DCT-MP is applied. A decrease in the top-K threshold corresponds to the activations

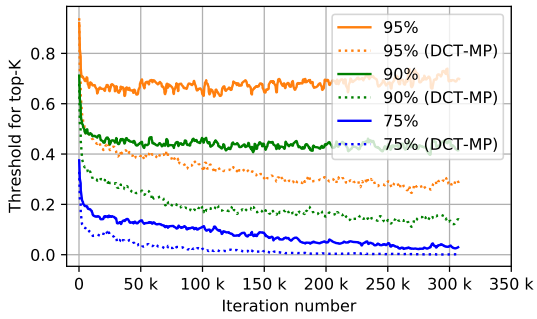
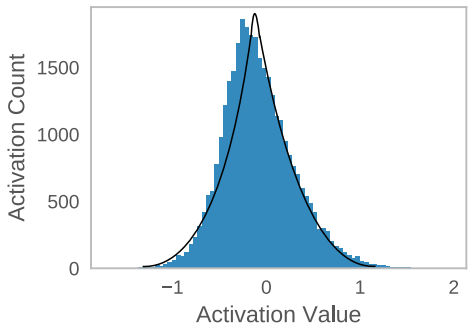
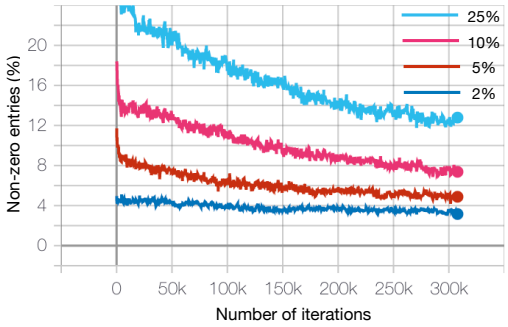


Figure 8.5: Top-K threshold for various levels of sparsity for the cases when compression using DCT-MP is applied and when it is not applied. The top-K thresholds decrease significantly when DCT-MP is applied. Thus, DCT-MP induces sparsity in neuron activations. This is possibly the reason for its improved generalization performance.



(a) Activation histogram after 10,000 iterations of training. The histogram approximately fits a Laplace distribution. This can be used to find an estimate of the sparsification threshold.



(b) Actual occupancy in X_{act} through the statistical estimates generated by assuming a Laplace distribution on entries (with target occupancy shown in legend).

Figure 8.6: Using Laplace distribution to model the entries of activation matrix and using it to predict the threshold values.

getting sparser (maybe approximately) as the training progresses. Thus, DCT-MP induces sparsity in activations while training, which is exploited for communication efficiency. An important advantage of such sparsity-inducing regularization is the improved generalization performance of the model, as shown in our experiments in Sec. 8.3. Our conjectured explanation for why sparsity helps in improving the generalization error is based on the performance of existing popular schemes. This includes dropout (see Fig. 8, [217]) and Rectified Linear Units (ReLU) (see Fig. 3, [220]), which themselves introduce sparsity in model activations, as well as implementations of implicit sparsity based methods in scalable algorithms for graph analysis [221], [222].

Reducing Compression Overhead through Statistical Estimates. DCT-MP requires $O(Bd \log d)$ time to sort each of the rows in $X_{act} \in \mathbb{R}^{B \times d}$ and find the corresponding threshold vector $\tau = [\tau_i]_{i=1}^{N}$. In some cases, this can contribute to an unacceptable computational overhead during the training

process. More importantly, it requires a similar sorting procedure during testing, which is not ideal since we want the model serving time to be low. In Figure 8.6a, we plot the histogram of activations after 10,000 iterations of training the DLRM model with the Criteo Ad Kaggle dataset (see Sec. 8.3.1 for details on the setup). We observe that the Laplace distribution closely approximates the distribution of entries in the activation matrix X_{act}.[1] Using this observation, we can estimate the threshold τ by finding the mean and variance of the Laplace random variable, say R, for that iteration using the activation matrix X_{act}. Thus, for a desired sparsity level η $(0 < \eta \leq 1)$, we can find the threshold τ as $\mathbb{P}(R \geq \tau) = \eta$ using the inverse distribution function (also called the quantile function) of the Laplace distribution.

An advantage of using such a statistical estimate for τ is that the compute time for compression reduces to $O(Bd)$ (to calculate the mean and variance). Further, during testing, the estimated τ from the training process can be used for sparsification, resulting in negligible compute overhead during serving.

In Table 8.1, we describe the train and test performance of the DLRM model with the Criteo Ad Kaggle dataset for DCT-MP with 2 workers. Here, the top-K threshold is found approximately by estimating it through the Laplace distribution fitting of entries of the activation matrix (see Fig. 8.6a for more details). We note that this scheme performs on-par with (or in some cases even better than) the exact top-K sparsification, while reducing the overhead of sorting both during training and inference.

Table 8.1: DCT-MP on the DLRM model with the Criteo Ad Kaggle dataset: Finding the top-K threshold using Laplacian estimation.

SPARSITY	TRAIN		TEST	
FACTOR	LOSS	ACC (%)	LOSS	ACC (%)
BASELINE	0.4477	79.23	0.4538	78.78
75%	0.4475	79.28	0.4529	78.84
90%	0.4473	79.31	0.4529	78.84
95%	0.4477	79.25	0.4528	78.82
98%	0.4477	79.23	0.4535	78.78

We observe that this scheme performs as well as top-K sparsification. However, a caveat of such a statistical scheme is that the desired sparsity level does not match the sparsity level obtained (see Fig. 8.6b). For example, when the desired sparsity level, η, is set to 75% (that is, 25% occupancy in X_{act}), the occupancy level varies throughout the training process, starting with 25% and ending with 13%. This is because even a slight deviation in threshold values results in huge changes in occupancy, especially for higher occupancy levels when the threshold is closer to zero. This might not be ideal when an exact sparsity level is desired to give guarantees on end-to-end

[1]The histogram in Fig. 8.6a is less well-fit by a Gaussian distribution, due to larger entries (not easily visible in Fig. 8.6a) because of the sparsity of activations, which is better modeled by the tail of the Laplace distribution.

training time. Although our initial results suggest that such a scheme performs as well as top-K sparsification, a more thorough investigation (which is beyond the scope of this dissertation) is required to demonstrate the practicality of this scheme.

Comparing Dropout and DCT-MP. Dropout and DCT-MP are similar in essence since they both selectively train neurons, but the two schemes are different: both in the goals they try to achieve, and in the mechanisms they use. Furthermore, they can be used complementarily. Here are the main differences between the two schemes. First, Dropout drops neurons randomly, while DCT-MP keeps only the most relevant neurons for each training sample. Second, for Dropout, going beyond 50% sparsity results in accuracy loss, but DCT-MP achieves up to 95% sparsification. Third, Dropout is applied to every parameter layer, but DCT-MP is applied only to the layers before the model split.

8.3 Empirical Results

In this section, we investigate DCT-MP and DCT-DP for three different experimental setups. In the first two subsections, we evaluate the performance of DCT-MP on the Deep Learning Recommendation Model (DLRM) and a Natural Language Processing (NLP) model for different levels of compression. We show that high compression factors can be obtained (up to $\sim 95\%$) with DCT-MP along with small improvements in model performance. We further evaluate DCT-DP on the DLRM model and see no loss in performance with up to 98% sparsity. Finally, we evaluate the performance of DCT-DP and DCT-MP on large-scale recommendation models that are trained using hybrid parallelism. We show that applying our algorithms can reduce the training time by as much as 37% for such large-scale models without any performance loss.

8.3.1 Experiments on the DLRM Model

Experimental Setup. For these experiments, we use the DLRM model from [195]. In this model, the dense features are first processed by a Multilayer Perceptron (MLP) with four layers, where each layer contains a Fully Connected (FC) layer followed by a Rectified Linear Unit (ReLU). Then, there is a feature interaction between the processed dense and sparse features, which goes through a second MLP with four layers (the last layer has Sigmoid instead of ReLU as the non-linearity) to produce the final output. In our experiments, the embedding dimension for sparse features was kept at 16, and the output dimensions of the four FC layers in the first MLP are 512, 256, 64 and 16, respectively. Similarly, for the second MLP, the output dimensions for the fours FC layers are 512, 256, 128 and 1, respectively.[2] Training and testing sets comprise of 6 days and one day, respectively, of the Criteo Ad Kaggle dataset.[3]

Fig. 8.7 provides an illustration of MP with the DLRM model. The shaded area in blue shows a sample partition for MP. In our simulations, we consider up to two splittings of the DLRM model.

[2]See the Criteo Kaggle benchmark for further details on the training process: https://github.com/facebookresearch/dlrm

[3]https://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/

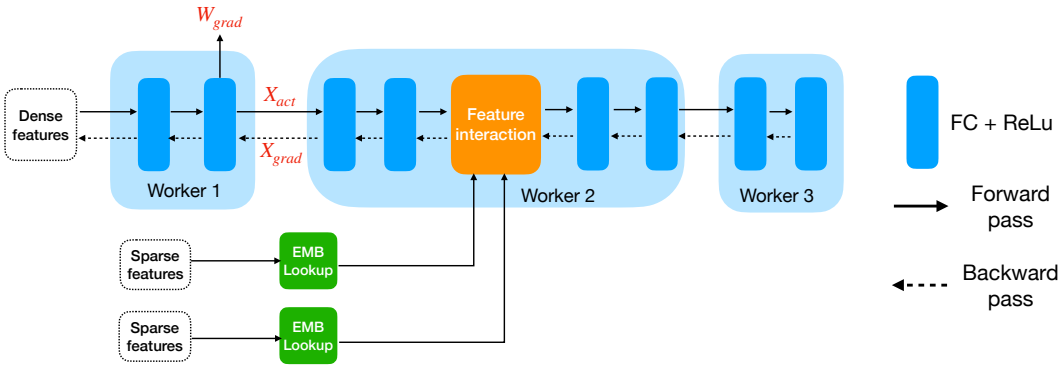


Figure 8.7: A illustration of model parallelism with DLRM. The entities that are sent across the network are shown in red. X_{act} and X_{grad} are communicated during MP, and W_{grad} is communicated during DP. The shaded area in blue represents a sample model partitioning for MP. In this case, three workers are working on one copy of the model during MP and comprise a trainer.

The first split is after two layers in the first MLP, and the second split is after two layers in the second MLP. Our goal is to reduce communication across different workers (both during the forward and backward passes). This is a typical setup in MP Training where workers 1, 2, and 3 can be the different pipes of a single trainer (e.g., see [199]). For all our experiments, the data shuffle remains constant across different training runs.

In Fig. 8.7, we mark the three entities that are sent across the network which we compress to alleviate communication costs in distributed DNN training. X_{act} and X_{grad} are the activation and gradient matrices sent across the network during the forward pass and backward passes, respectively. The third entity that can be compressed is the parameter gradient (shown as W_{grad}) that is sent from Workers 1, 2, and 3 to the parameter server. This keeps a central copy of weights and updates it regularly through the gradients received from different workers.

In Table 8.2, we show the cross-entropy loss [223] and accuracy with the DLRM model on the training and testing data samples. A sparsity factor (η) of 0% denotes the baseline with no compression. We consider two settings for MP: one split (that is, 2 MP workers); and two splits (or three workers for MP).

MP with two workers (one split). In rows 2-5 in Table 8.2, we consider one split in the model (or MP with two workers) in the first MLP after two layers. We see that even with 95% sparsity (that is, $20\times$ compression) on X_{act} (and X_{grad}) sent across the network, we are able to perform better than baseline (with no compression), both in terms of train and test loss (highlighted in bold cases). However, we see a tangible loss in performance when the sparsity is further increased to 98%.

MP with three workers (two splits). In rows 6-8 in Table 8.2, we consider MP with 3 workers, where the two model splits are in the first and second MLP, as shown in Fig. 8.7. Note that, in the case of two splits, compressing the entities that are sent across the network by up to 90% does not affect the test accuracy, and it is still better than the baseline with no compression. However, increasing the sparsity factor to 95% is too ambitious for the two split case, and it increases the test loss by 0.18%. Further increasing the number of splits results in a greater performance loss, and the

Table 8.2: DCT-MP on the DLRM model: Train and Test Loss and Accuracy for multiple sparsity ratios (denoted by η) and different settings for MP.

η	MP Workers	Train Loss	Train Acc (%)	Test Loss	Test Acc (%)
0%	–	0.4477	79.23	0.4538	78.78
75%	2	0.4473	79.29	0.4532	78.81
90%	2	0.4472	79.28	0.4530	78.81
95%	2	0.4473	79.24	**0.4534**	78.80
98%	2	0.4505	79.07	0.4562	78.61
75%	3	0.4482	79.19	0.4536	78.79
90%	3	0.4479	79.24	**0.4537**	78.78
95%	3	0.4495	79.18	0.4546	78.72

performance is worse than baseline for even 75% sparsity.

Remark 12. We emphasize that for all the experiments in this paper, the location of splits for MP were not tuned as hyperparameters. Instead, we inserted splits after randomly chosen FC layers, or after the ReLU following the FC layer if it exists. The advantage of inserting a split after ReLU layers is that the activation matrix is 50% sparse on average, resulting in higher compression rates for DCT-MP.

Table 8.3: DCT-DP on the DLRM model: Train and Test Loss and Accuracy for various levels of sparsity.

Sparsity Factor	Train Loss	Train Acc (%)	Test Loss	Test Acc (%)
Baseline	0.4477	79.23	0.4538	78.78
75%	0.4478	79.23	0.4534	78.81
90%	0.4478	79.22	0.4536	78.79
95%	0.4479	79.25	0.4538	78.79
98%	0.4478	79.23	**0.4537**	78.80
99.5%	0.4482	79.20	0.4547	78.75

DP with the DLRM Model. In Table 8.3, we illustrate the performance of DCT-DP on DLRM by compressing the gradients of the parameters of all the 8 FC layers while they are sent across the wire to the parameter server. The parameter server then updates the model parameters using the compressed gradient. We use error feedback [91] to compensate for the error in gradient compression by feeding it back to the gradients in the next iteration. In general, DCT-DP compression enjoy

higher compression rates due to the use of error compensation schemes and the fact that error in one layer does not propagate to the other layers, unlike in the case of MP compression. Compression up to 98% sparsity does not show any loss in performance. However, further compressing to 99.5% sparsity increases the test loss by 0.20%.

Table 8.4: Compression using DCT-DP and DCT-MP on the DLRM model: Train and Test Loss and Accuracy with two MP splits (that is, three workers for MP).

SPARSITY FACTOR	TRAIN LOSS	TRAIN ACC (%)	TEST LOSS	TEST ACC (%)
BASELINE	0.4477	79.23	0.4538	78.78
75%	0.4480	79.23	0.4535	78.81
90%	0.4481	79.26	**0.4537**	78.78
95%	0.4492	79.19	0.4548	78.70

Communication-efficient Hybrid Training. Next, we apply compression to W_{grad} for the 8 FC layers (in the DP case) and to X_{act} (and X_{grad}) for two splits (in the MP case) and present our results in Table 8.4. We see that compression up to 90% sparsity (both during DP and MP) does not affect the performance, but the test loss increases by 0.22% when the sparsity factor is increased to 95%.

8.3.2 Experiments on a Translation Model

For our experiments with DCT-MP, we next consider the Transformer translation model as an application of NLP using DNNs. We train over the IWSLT'14 German to English dataset [224]. The setup and hyperparameters were directly borrowed from the fairseq NLP Library [225]. The model used was borrowed from [226], where both encoder and decoder have 6 layers, each of which uses a fully connected Feed-Forward Network (FFN) with input and output dimensionality of 512 and inner layer dimensionality of 1024.[4] We report the training and testing losses and the BLEU scores after 50 epochs of training.

Our results with DCT-MP on the translation model are described in Table 8.5. We consider three training scenarios: Two MP workers (with one split), Three MP workers (with two splits), and Five MP workers (with 4 splits). For the case with one split, we inserted the DCT-MP operator after the ReLu operator in the FFN of the fifth encoder layer. For the two splits case, we additionally inserted the DCT-MP operator after the ReLu operator in the FFN of the fifth encoder layer. We further added two splits after the ReLu operator in the third FFN in both the encoder and decoder layers for the four splits case. For each scenario, we show the best performing sparsity factor in bold.

[4]For further details on the translation model, dataset preprocessing and the hyperparameters used, see https://github.com/pytorch/fairseq/tree/master/examples/translation

We emphasize that no hyperparameter tuning was performed in choosing the splits, and we observed in our experiments that using DCT-MP after an FC Layer or a ReLu layer improves the generalization performance, possibly due to (implicitly) added regularization (as illustrated in Fig. 8.5). Note that we can add more MP splits for the NLP model compared to the DLRM model since the model is significantly deeper (and thus less susceptible to changes in outputs of a few layers) with larger FC layers (thus allowing for greater sparsity). This shows that DCT-MP is more beneficial for wider and/or deeper models (that is, typical setups where MP is used).

Table 8.5: DCT-MP on a translation model with IWSLT'14 dataset: Train and Test Losses and BLEU scores for various levels of sparsity and different splits for MP.

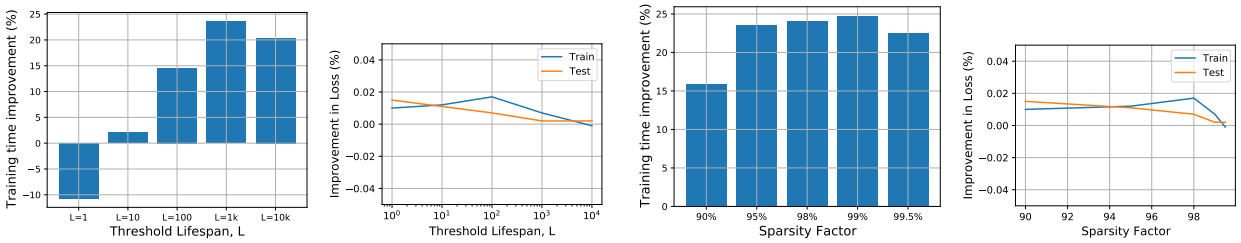
SPARSITY FACTOR	MP WORKERS	TRAIN LOSS	TEST LOSS	BLEU SCORE
BASELINE	–	3.150	3.883	35.17
90%	2	3.159	3.879	35.23
95%	2	3.157	**3.882**	35.18
90%	3	3.151	3.881	35.22
95%	3	3.148	**3.882**	35.19
90%	5	3.157	**3.882**	35.20
95%	5	3.188	3.890	35.15

In this subsection, we do not consider DCT-DP since similar schemes have been evaluated for NLP models in existing works such as [88] and [95]. In the next subsection, we evaluate DCT-MP and DCT-DP on large-scale recommendation models for end-to-end training times and overall model performance.

8.3.3 Large-Scale Recommendation Systems

We present our results for a real-world large scale recommendation system that employs HP for parallelization on click-through rate prediction task. We employ DCT-MP and DCT-DP to reduce the network bandwidth usage in these systems.

Experimental Setup. We leverage a distributed data-parallel asynchronous training system with multiple trainers to train a recommendation model. Each trainer in the DP setup may consist of one or more workers that use MP (see Fig. 8.1 for an illustration). Typically, the model is split into 10 or more parts and fine-grained parallelism is employed for high throughput. Hence, the worker machines suffer from very high communication cost for both MP and DP. The batch sizes are usually in the range of 100-1000, but they are employed with hogwild threads (see [227]) to increase the throughput of the system, further exacerbating the communication cost problem. The recommendation model considered in this section takes multiple days to train with general-purpose CPU machines. All the workers and parameter servers run on Intel 18-core 2GHz processors



(a) Training time improvements for different values of L.

(b) Train and test error improvements for different values of L.

(c) Training time improvements for various sparsity factors.

(d) Train and test error for various sparsity factors.

Figure 8.8: DCT-DP on Large-Scale Recommendation Models. Figures (a) and (b) show the training time and loss improvements, respectively, over baseline for different values of the threshold life-span, L, for a sparsity level of 95%. Figures (c) and (d) show the same statistics for various levels of sparsity for $L = 1000$.

with 12.5Gbit Ethernet. The hardware configurations are identical and consistent across all the experiments. We train using 7B training examples and evaluate the model on 0.5B examples. For quantifying model performance, we report the cross-entropy loss from the classification task. We compare relative cross-entropy loss and end-to-end training times of the proposed techniques with respect to a baseline model without communication compression.

DCT-DP with Large-Scale Recommendation Model. Figure 8.8 shows the results of applying DCT-DP on the large-scale recommendation model. In Figure 8.8a, we plot the improvements in end-to-end training times when DCT-MP is applied to compress the parameter gradients, W_{grad}, that are sent to the parameter server. Here, we keep the sparsity level constant at 95% and vary the threshold life-span L (the interval after which the top-K threshold is updated). We note that compression with $L = 1$ takes 11% more time than the baseline with no compression. This is due to the cost of the copy-and-sort routine which computes the top-K threshold. Increasing L to 1000 trains the model 23% faster and further increasing it to 10000 does not provide any additional gain. Figure 8.8b illustrates that for different values of L, the train and test losses are within 0.01% of the baseline performance.

Fig. 8.8c shows the improvement in training time for various levels of sparsity when the threshold life span is kept constant at $L = 1000$. We observe the general trend that when the sparsity is increased, the training time improves. Overall, we are able to compress the gradients to sparsity factors of up to 99.5% without any loss in train and test performance (as noted from Fig. 8.8d). However, we do not see significant improvements in training time beyond the sparsity level of 95%, possibly because the message size is small enough to not hurt bandwidth usage, and the only cost remaining is the fixed latency cost associated with sending any message, irrespective of its size. Further, we observe that error feedback works very well in this asynchronous data-parallel training paradigm with a larger number of hogwild threads.[5] We share the error feedback buffer between the

[5]We note that the existing works prove convergence guarantees only for the synchronous SGD settings, but we see that error feedback works well even in the asynchronous setting.

Table 8.6: DCT-MP on a large-scale recommender model

SPARSITY FACTOR	LOSS IMPROVEMENT (%) TRAIN	TEST	TIME GAIN (%)
BASELINE	0.000%	0.000%	0.000%
75%	-0.006%	0.023%	7.04%
90%	-0.021%	**0.016%**	**13.95%**
95%	-0.070%	-0.121%	14.43%

multiple threads, and hence the magnitude of error in the buffer can grow quite fast leading to stale updates. To avoid this, we drain the error feedback buffer stochastically every $1M$ iterations.

DCT-MP with Large-Scale Recommendation Model. We employ DCT-MP to compress the entities sent through the network during MP for communication efficiency. DCT-MP is applied across the 12 splits of the model after the ReLU layer. Our results are summarized in Table 8.6. We show improvement in training and test losses[6] in columns 2 and 3, respectively, and the improvements in end-to-end training times in column 4 for various levels of sparsity. We observe that the training performance slightly degrades with DCT-MP on large-scale models. However, the test performance improves up to sparsity levels of 90%, with a 14% improvement in end-to-end training time. Increasing the sparsity level to 95% degrades the test performance by 0.121%. Note that we can further improve the performance of DCT-MP by identifying the layers whose activations are sensitive to sparsification and avoiding compressing them during DCT-MP (or changing the location of the split). However, such selectivity in choosing layers for DCT-MP is beyond the scope of this paper.

Communication-Efficient Hybrid training. Next, we apply both DCT-DP and DCT-MP for communication reduction during hybrid training of a large-scale recommendation model. Inspired by our previous results, we chose the sparsity levels as 90% and 99% for DCT-MP and DCT-DP (with $L = 1000$), respectively. We observe a 37.1% reduction in end-to-end training time, with train and test loss within 0.01% of the baseline model that does no compression.

Further, before applying DCT, we observed that the network utilization was high (94.2%) and the CPU utilization was low (48.7%), implying that communication is a bottleneck. However, after applying DCT, CPU utilization increased to 91.1% and network utilization decreased to 49.3%, implying that DCT shifted the bottleneck from communication to computation. In general, metrics like CPU and network utilization can be used to check if DCT can help in training by reducing communication (e.g., in the case of network bandwidths higher that 12.5 Gbps considered in this paper).

[6]Positive numbers imply better performance.

8.4 Compression Schemes That Do Not Work

We saw in Sec. 8.3 that activations during the forward pass and gradients during the backward pass can be compressed by large factors (up to $20\times$) using DCT-MP. This is due to selecting and training only the most relevant neurons corresponding to a given training sample. In this section, we present some negative results with other methods to compress the activation and gradient matrices during the forward and backward passes, respectively.

Gaussian Sketching for Activation Compression. Here, we use a Gaussian sketching scheme to compress the activations going forward. In Randomized Numerical Linear Algebra (RandNLA), the idea of sketching is to represent a large matrix by a smaller proxy that can be further used for matrix operations such as matrix multiplication, least squares regression, and low-rank approximation [41], [47], [228]. The sketched version of a matrix A is given by $A \times S$, where S is a random sketching matrix (e.g., all entries of S are sampled i.i.d. from an appropriately scaled Gaussian distribution).

In Table 8.7, we compress the activations during the forward pass using Gaussian sketching. Unlike the DCT-MP algorithm, we do not compress the gradients during the backward pass. The aim is to identify if a low-rank structure exists in the activation matrix that can be used to compress the activation matrix in general.

Table 8.7: Compressing the activation matrix during MP using Gaussian sketching does not yield good results.

Compression Factor	Train		Test	
	Loss	Acc (%)	Loss	Acc (%)
Baseline	0.4477	79.23	0.4538	78.78
50%	0.4569	78.72	0.4618	78.37
75%	0.4610	78.53	0.4656	78.12
90%	0.4685	77.95	0.4721	77.78

As seen in Table 8.7, sketching techniques directly borrowed from RandNLA do not perform as well. This is likely because such schemes were designed to cater to operations such as low-rank approximation, where the matrices to be compressed are generally well-approximated by low-rank matrices. For instance, Gaussian sketching has seen success in approximate least squares regression and low-rank matrix approximation [41], [47], [228]. This suggests that the activation matrix for DNNs, in general, does not reside in a subspace that is sufficiently low-rank to be meaningfully used for compression.

Top-K Thresholding for Gradient Compression. We saw in Sec. 8.3 that the parameter gradients (illustrated as W_{grad} in Fig. 8.1) can be compressed to high factors with any loss in accuracy when used with appropriate error compensation. However, the same is not true for the gradients with respect to hidden neurons (illustrated as X_{grad} in Fig. 8.1) that are sent across the network during the backward pass in MP. This can be seen from our results in Table 8.8, where we

apply gradient compression using top-K thresholding with error feedback. Further, we observed that training without error feedback can cause divergence.

Table 8.8: Compressing the gradient matrix during backward pass in MP using top-K sparsification does not yield good results.

COMPRESSION FACTOR	TRAIN		TEST	
	LOSS	ACC (%)	LOSS	ACC (%)
BASELINE	0.4477	79.23	0.4538	78.78
50%	0.4495	79.07	0.4561	78.62
75%	0.4516	78.95	0.4588	78.48
90%	0.4701	77.76	0.4789	77.13

Our hypothesis on why compressing the gradients of the hidden neurons by top-K thresholding does not yield good results is due to the propagation of error to the initial layers. Consider the following example to illustrate this. Consider the following deep network, where we have several vector-valued functions $A(\cdot), B(\cdot), C(\cdot), \cdots, L(\cdot)$ composed in a chain, that is $A \rightarrow B \rightarrow C \rightarrow \cdots \rightarrow K \rightarrow L$. Algebraically, the loss looks like $L(A) = L(K(\cdots C(B(A)) \cdots))$. Then, the gradient of the loss with respect to A is given by the multiplication of the Jacobians, that is, $J_L(A) = J_L(K) \times \cdots \times J_C(B) \times J_B(A)$. (Here, $J_L(A)$ denotes the gradient of L with respect to A.) If we change any of the Jacobians in between (that is, compress the gradient X_{grad} with respect to hidden neurons), then the error is propagated all the way to the initial layers of the network. Even adding error feedback to the compression process does not recover the lost accuracy.

8.5 Conclusions

Machine learning models are consistently growing in size with more and more compute nodes used for training. Inspired by the fact that communication is increasingly becoming the bottleneck for large-scale training, we proposed two practical algorithms, DCT-DP and DCT-MP, to reduce the communication bottleneck during data and model parallelism, respectively. DCT-DP and DCT-MP improve end-to-end training time by sparsifying the matrices to be sent across the wire by appropriately selecting a sparsification threshold. We observed that using the proposed algorithms for hybrid training of DNNs can reduce the end-to-end training time by up to 37%. Further, using DCT-MP during model parallelism induces sparsity in hidden activations which helps in reducing generalization error.

Part IV

Resource Allocation and Pricing

In this part of the thesis, we study pricing and resource allocation for serverless computing:

- **Chapter 9:** From the Expected Utility Theory Perspective
- **Chapter 10:** From the Cumulative Prospect Theory Perspective

Chapter 9

Expected Utility Theory Perspective

Serverless computing platforms currently rely on basic pricing schemes that are static and do not reflect customer feedback, which leads to significant inefficiencies from a total utility perspective. In this chapter, we propose a novel scheduler to allocate resources for serverless computing with the help of utility functions to model the delay-sensitivity of customers.

9.1 Introduction

In this chapter, we provide a guide to resource allocation in next-generation cloud computing systems. The pricing mechanism (and the corresponding resource allocation scheme employed by the service provider) is one of the most important tools in influencing the usage of cloud resources. A typical customer's goal is to obtain the highest quality of service (QoS) for a reasonable and affordable price. Thus, how the service provider allocates resources and charges its customers affects customer behavior, loyalty to the provider, and ultimately its success. Current popular cloud computing providers–such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud–employ a pricing scheme referred to as "pay-per-use fixed pricing." It scales linearly in the resources utilized, such as time, memory, and the number of jobs, regardless of the nature (e.g., delay-sensitive or not?), and importance (critical or not?) of the users' applications. There are several reasons to revisit this arcane pricing scheme philosophy, as we discuss below.

Cloud computing is a new ecosystem with a growing customer base. Currently, cloud providers can schedule most of the users' jobs instantly as the resources exceed the demand [229]. However, this surplus supply luxury is neither sustainable in the long term with the growing demand of a large customer base seeking cloud computing services, nor is it economically viable considering alternatives such as we address in this paper. Hence, it is extremely desirable that we have a market-based pricing scheme that adapts prices to the demand, and allocates resources to the users that need them the most. (See [105] and the references therein for a comparison between different pricing models and schemes that are either currently being employed or that have been proposed with simulations.)

In this paper, we focus on a recent cloud service called *serverless computing* that has garnered

significant attention from industry (e.g., Amazon Web Services (AWS) Lambda, Microsoft Azure Functions, Google Cloud Functions) as well as the systems community (see, e.g., [1]–[4], [7]). Serverless computing is especially appealing due to the *elasticity* of the serverless system, where the typical jobs submitted are comprised of simple functions that have relatively small system requirements such as execution time and memory storage. Besides, these functions are stateless in the sense that the function state is kept in storage and hence these functions can be viewed as standalone with minimum dependence on other functions [1], [7]. This allows us to treat each user's job as comprised of several independent unit functions that can be executed on a single serverless worker, thus simplifying our model.

Another important feature of serverless systems is the elimination of up-front commitment by users. The jobs are typically triggered by external events (e.g., receipt of a message); see [229] for examples of different triggers. However, as a result, the demand for resources can vary significantly over time and it is important to allow the prices to adapt to this changing demand in real-time. In particular, this will ensure that we serve the users who most value their jobs during surge periods, and are therefore willing to pay premium prices.

Pricing based on delay-sensitivity: In addition to dynamic pricing, it is important to differentiate pricing for jobs based on their *delay-sensitivity* and allocate resources accordingly. Job completion times form an integral part on service level agreements that governs the quality of service of the users [230], [231]. Users have heterogeneous jobs and have different requirements with respect to their service delay. For example, *urgent* jobs (that need to be executed in real-time, such as model deployment [12] and real-time video compression [20]) may need prioritization, whereas *enduring* jobs (that can be put into queues with reasonable wait-times, such as optimization in machine learning [6], [14], [16], [21] and scientific computing [3], [5], [8]) could be put on hold. It is therefore desirable for the pricing scheme to provide appropriate incentives to users. For example, premium rates could be applied to urgent jobs, and discounts applied to enduring jobs.

To address this issue, we develop a *dynamic multi-tier pricing scheme* that incentivizes users to bid optimally for resources that are tailored to their requirements and delay-sensitivity characteristics. To articulate the notion of demand and delay-sensitivity, we adopt the concept of utility functions from economics, commonly used to measure user preferences over a set of goods and services. We consider user utilities as a function of delays in job completion times. This enables us to naturally differentiate jobs based on their delay-sensitivity characteristics and allocate the resources optimally. Some examples of such utility functions are shown in Fig. 9.1 for three users. User 1 obtains utility only when her job gets completed under 0.1 seconds. User 2 obtains diminishing returns as time passes. User 3, on the other hand, does not care as long as her job is completed within 10000 seconds.

We set for ourselves the goal of scheduling jobs so that the net utility gained by the users is maximized. We achieve this by formulating and solving an optimization problem that maximizes the social welfare, i.e., the sum of the utilities received by all the users. Social welfare as a concept is very important to many companies like Amazon that prioritize customer satisfaction[1]. A prototype for our scheduler is shown in Figure 9.2. It takes as its input the job sizes and the utility functions

[1]For example, see https://www.amazon.jobs/en/principles

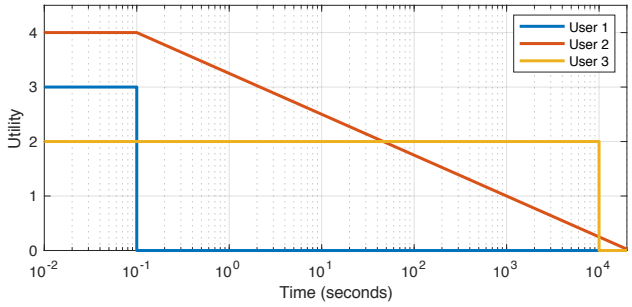


Figure 9.1: Examples of utility functions for 3 users that depend on completion time.

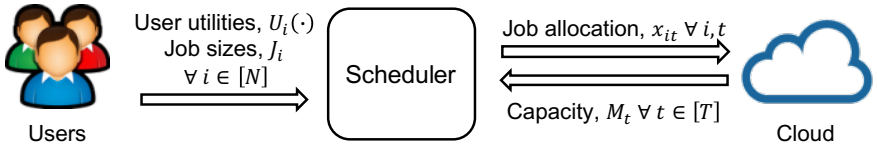


Figure 9.2: A block diagram schematic of the system with N users and T service tiers

for each user, the capacity constraints or machine availability information from the cloud, and outputs a job allocation schedule. Here, the job sizes capture the demand for resources, the utility functions capture the delay-sensitivity of the users, and the capacity constraints capture the supply of resources.

The feature of pricing cloud resources based on delay-sensitivity of users has been lacking from most of the current pricing schemes that are being implemented as well as several real-time dynamic pricing schemes that have been proposed in the literature (we refer the readers to [105], [106] and [107] for surveys on existing and proposed pricing schemes for the cloud). Notable exceptions are [232]–[234], where job completion deadlines affect pricing. Our problem formulation is closest to the model considered in [232]. However, we take a different approach towards solving it leading to a widely different pricing and allocation scheme. (We elaborate on this in Remark 15.) In [232], it is assumed that the service provider knows the users' utility functions and each user is charged an amount equal to the utility corresponding to her delay in service. However, the service provider does not have access to the users' utility function in practice and is one of the primary difficulties in implementing schemes that assume this knowledge. Indeed, a utility function is an abstract concept used to capture the delay-sensitivity of the users and oftentimes the users themselves are unaware of their utility functions.

It is common to consider the relatively more tangible notion of *willingness to pay* in lieu of the utility function. For example, if a user is willing to pay a maximum of $\$1$ for her job to get completed with a delay of 1 minute, then we will say that her utility function takes value $\$1$ at $t = 1$ minute. In Fig. 9.1, user 1 is willing to pay more than user 3 to get her job completed within 0.1 seconds.

Certainly, the service provider still needs to procure this willingness to pay information from the users in a truthful manner. We propose a pricing scheme that ensures that the users cannot gain

by faking their willingness to pay. In our proposed pricing scheme the users are not necessarily charged by the full amount of their willingness to pay. Instead, the amount charged depends on the net demand and the willingness to pay of all the users with the additional feature that no user under any circumstances is charged more than her willingness to pay.[2]

Utility-agnostic approach: In general, the utility of a user is a function of time and could be hard to communicate. Even if we consider a family of typical utility functions, which could potentially resolve the problem of communication, there exist privacy concerns under which users are reluctant to report their utility functions. To mitigate this, inspired by the seminal works in [236], [237], we decompose the scheduler optimization problem into user problems—one for each user—and a cloud problem. Each user problem interacts with the cloud problem through the prices published by the cloud provider and the corresponding budget responses from the users. We prove that under equilibrium conditions, the above decomposition solves the original scheduler optimization problem. However, in this framework the users need to report only their optimal budget responses which is a finite vector of the size of the number of service tiers offered. A user's budget response is a function of her utility function and the prices published by the cloud. This inserts a filter between the utility functions of the users and their reports to the network providing them a layer of privacy and making the communication between the cloud and the users feasible. Thus, the above decomposition allows us to track the optimal performance with limited feedback from users in the form of their budget responses. We demonstrate the viability of our approach through simulations.

In situations where it is infeasible to get budget responses from the users at a desired response rate in order to run the optimization algorithm, it is possible to emulate it as follows: locally, on the user's end, we let each user choose her willingness to pay from a list of options that best suits her needs. Using this willingness to pay, a local algorithm computes the optimal budget response to the prices published by the user and transfers only the budget signals to the cloud keeping the willingness to pay information encrypted locally. Besides, an attractive feature of such a scheme is that the users can update their willingness to pay at their disposal and the local algorithm will use the most recent choice (see Fig. 9.4 for an illustration). Our algorithm can be interpreted as a protocol between the users and the server for optimal resource allocation and fair pricing (further explained in Sec. 9.4). This can be viewed as the scheduling analog of the window-based bandwidth allocation method in TCP protocol that has revolutionized the field of network resource allocation [238]–[240].

Our contributions and organization: To the best of our knowledge, this is the first paper to study resource allocation and pricing for serverless systems that takes into account users' delay-sensitivity and allocates resources via a utility-agnostic approach.

In section 9.2, we describe our novel problem formulation for scheduling jobs in a serverless system. We formulate our scheduling problem as an optimization problem to maximize the net social welfare, i.e. the sum of the utilities of all the users. We note that the formulated optimization

[2]This is similar in spirit to the dynamic pricing schemes currently deployed in service providers such as AWS spot pricing for EC2, where the user defines the maximum price she is willing to pay. However, we take into account the market fluctuations and user preferences such as delay-sensitivity unlike the naive schemes currently implemented [235].

problem is NP-hard in general and consider an LP-relaxation of this problem instead. We prove that the relaxed solution is close to the original NP-hard problem. In particular, we provide an upper bound on the gap between the original optimization problem and its relaxation (section 9.3). We then decompose our relaxed optimization problem into multiple user problems–one for each user–and a cloud problem. This gives rise to equilibrium prices for each tier corresponding to the optimal resource allocation. In section 9.4, using the ideas developed in the previous two sections, we propose an algorithm to allocate resources across different tiers in the subsequent runs of the scheduler and give a gradient-based pricing scheme to track the equilibrium prices.

In section 9.5, we give a market simulation where the users' utilities change every day. We compare our allocation scheme with existing schemes (such as first-come-first-serve) that do not account for users' utilities. We observe that the sum utility of the system obtained by using our allocation scheme almost doubles the sum utility obtained otherwise. Further, we show that our utility-agnostic approach successfully tracks the variations in users' utilities across days.

We conclude our paper in section 9.6 with a few exciting open problems of interest to the research community like machine heterogeneity and job dependencies, and other important considerations for pricing like wholesale discount and user risk-preferences and fault-tolerance. We give a brief survey of related works in Appendix 9.7. To maintain continuity in reading, we defer the proofs and additional remarks to Appendices 9.8 and 9.9, respectively.

9.2 Problem Formulation

Serverless (also called Function-as-a-service) as a cloud computing framework is ideal for "simple" jobs where each *user* submits a *function* to be executed on serverless workers. Each user can request for a job comprising of any number of executions of her function at any time, which could be triggered due to external events. For example, the users can provide the conditions under which they require the execution of a certain number of instances of their function. The users provide these *trigger event* details along with their function submissions. Our goal here is to design an efficient real-time job scheduler to allocate resources to the jobs that have been triggered across multiple users and a corresponding pricing scheme for the cloud provider.

We envision a job scheduler that operates periodically (say every 0.1 seconds) and schedules the jobs that are currently in its queue. This queue consists of all the jobs that have been triggered and are ready to be executed but have not been scheduled yet. Thus, it consists of previously unscheduled jobs (complete or partial) plus any new jobs that arrived since the previous scheduling run. We assume that each function execution requires one serverless worker and takes unit time (e.g., 100 ms). Also, a user derives utility only when her job is completed, that is, when all the functions comprising her job are executed. Thus, we focus on the case where user utilities are only a function of job completion times, and let the respective delay-sensitivity for user i be captured by a utility function $U_i : [0, \infty) \to \mathbb{R}$. That is, user i obtains utility $U_i(\tau)$ if her job is completed at time instant $\tau(> 0)$, where $U_i(\cdot)$ is non-increasing. Let J_i denote the number of function executions needed to complete the job of user i. We call this the size of user i's job. For example, it could be a single function instance in which case $J_i = 1$ or a batch job of size $J_i > 1$.

We think of the scheduler as allocating resources to the users in different *service tiers* based on their execution times—some jobs will be scheduled for immediate execution, whereas others will be scheduled for execution at later times in the future. The jobs that are not scheduled will remain in the queue to be scheduled by the scheduler at later operations. Note that the jobs that have been scheduled for execution at a future point are removed from the queue. Let T be the maximum number of service tiers offered by the cloud provider, and let the t-th tier be characterized by the end time of this tier given by τ_t. For example, if $T = 5$, then the five tiers can correspond to job completion under 1 second, 10 seconds, 10 minutes, 1 hour and 10 hours, respectively. Notice that the end times of the different tiers in our model need not be evenly spaced. This is a useful feature because it allows our scheduler to plan over longer time horizons with a limited number of tiers. The pricing for these different tiers of service should ideally decrease as the job completion time increases.

We refer to the intervening time between the adjacent tier end times as *service intervals*. Continuing the previous example, service interval 1 starts at 0 and ends at 1 second, service interval 2 starts at 1 second and ends at 10 seconds, etc. Note that the functions in a single user's job can be served across several service intervals, but has a unique tier when all these functions are completed. Further, let M_t be the constraint on the number of machines available for the scheduler to allocate resources in service interval t. For example, if 500 machines are available every 100ms and tier 1 is one second, then $M_1 = 500 * (1/0.1) = 5000$. Similarly, if tier 2 is 10 seconds, then the number of machines available for tier 2 jobs are $500 * (10/0.1) - 5000 = 45,000$. However, we would like to retain some portion of this for the subsequent operations of our scheduler. This will allow the scheduler to allocate resources for urgent jobs arriving in the future. Say we decide to utilize at most 20% of these machines, then $M_2 = 45,000 * (0.2) = 9000$. More generally, M_t would depend on several factors like machine occupancy by pre-scheduled jobs, retained machines for future scheduling, a margin to account for system uncertainties, etc.

System Problem: At each implementation of the scheduler, we assume that the scheduler has access to the unscheduled jobs ready to be executed (say job of size J_i for user i), their utility functions adjusted for past delay (for example, if a job has been waiting in queue for time τ_0, then we take its utility function to be $U_i(\tau + \tau_0)$), and the machine availability M_t for each service interval t. The scheduler then outputs a feasible allocation of serverless workers across different service intervals. Let $x_{i,t}$ denote the number of functions executed for agent $i \in [N] := \{1, 2, \ldots, N\}$ at service interval $t \in [T] := \{1, 2, \ldots, T\}$. Let $\mathbf{x}$ be the $(N \times T)$-matrix with entries $x_{i,t}$, $i \in [N], t \in [T]$.

Let $U_{i,t} := U_i(\tau_t)$ denote user i's utility if her job is completed in tier t, where τ_t is the end time of tier t. Then

$$T_i := \min\{t \in [T] : \sum_{s=1}^{t} x_{i,s} \geq J_i\}, \tag{9.1}$$

is the time it takes to complete user i's J_i functions, awarding her a utility of U_{i,T_i}. (If a user's job is not completed we let $T_i = T + 1$ and assign her zero utility.) Also, at any service interval $t \in [T]$, since the number of function executions cannot exceed the cloud service provider's capacity, we have $\sum_{i=1}^{N} x_{i,t} \leq M_t$. To this end, we formulate the system problem SYS that maximizes the sum

utility of the system as follows (see Fig. 9.2 for an illustration):

$$\underline{\textbf{SYS}}$$

$$\underset{x_{i,t}\geq 0}{\text{Maximize}} \quad \sum_{i=1}^{n} U_{i,T_i}$$

$$\text{subject to} \quad \sum_{i=1}^{N} x_{i,t} \leq M_t, \forall\, t \in [T], \text{ and} \tag{9.2}$$

$$T_i = \min\{t \in [T] : \sum_{s=1}^{t} x_{i,s} \geq J_i\},\ \forall\, i \in [N]. \tag{9.3}$$

In general, SYS can have multiple solutions, but there always exists a solution that assumes a special form. Specifically, it corresponds to a *non-preemptive scheduling*, where the resources are allocated so that none of the users' jobs are interrupted in the middle of the execution and made to wait till its execution resumes at a later stage after serving other users. We first outline a rough sketch of the underlying intuition and then establish this result in Lemma 14. (See [241] for a similar result in the context of scheduling jobs on a single machine.) Since the utilities of users depend only on their job completion times, it is suboptimal to leave a user hanging by allocating partial resources (which provides no utility gain). For example, say user i has been allocated a total of $\sum_{s=1}^{t} x_{i,s} (< J_i)$ machines till the end of service interval t, for some $1 \leq t < T$, and then interrupted, i.e. $x_{i,(t+1)} = 0$. Further, let there be a user $j (\neq i)$ that is allocated resources in the service interval $t+1$. If user i's job completes before user j, then we can swap the function executions of user i at times $> t+1$ with the foremost function executions of user j without reducing the sum utility, since such a swap will only possibly reduce the completion time of user i without affecting that of user j. On the other hand, if user j's job completes before user i, then we can swap the function executions of user i at times $\leq t$ with the hindmost function executions of user j without reducing the sum utility, since such a swap will only reduce the completion time of user j without affecting that of user i.

Corresponding to any non-preemptive scheduling, there exists an implicitly defined priority ordering amongst the users based on their completion times. Without loss of generality, suppose that this ordering is $1 \leftarrow 2 \leftarrow 3 \leftarrow \cdots \leftarrow N$, that is, user i is served along or before user $i+1$ for all $i \in [N-1]$. A simple greedy allocation $\tilde{\mathbf{x}} := (\tilde{x}_{i,t})_{i\in[N],t\in[T]}$ corresponding to this priority ordering would work as follows: Say the cloud has served the first $i-1$ users till time t. Now, if the number of machines remaining at time t is greater than J_i, all of user i's functions are allocated at time t. Otherwise, the cloud provider continues allocating resources at times $> t$ to user i till her job is complete before attending the user $i+1$. This process continues till either all users or all of the T tiers are served. (See remark 13.)

Lemma 14 (Non-preemptive scheduling)**.** *There exists an optimal solution to SYS that allocates resources to users in an uninterrupted fashion, that is, if user i gets allocated some resources, the system would allocate J_i resources to user i (possibly across multiple service intervals) instead of halting it and attending other users.*

Let $u_{i,t} := U_{i,t} - U_{i,t+1},\ \forall i, t$, where we assume that $U_{i,T+1} := 0, \forall i$, consistent with the fact that we assign zero utility if the jobs are not completed. Thus, $u_{i,t} \geq 0, \forall i, t$, as $U_i(\cdot)$'s are monotonically decreasing. Condition (9.3) in problem SYS is not favorable from an optimization framework point of view. Hence, we introduce an indicator variable $y_{i,t}, \forall i, t$, that we use as a proxy to indicate whether user i's job is completed on or before time t. Let $\mathbf{y} := (y_{i,t})_{i\in[N],t\in[T]}$. With these variables, the scheduler optimization can be formulated as follows:

SYS-ILP

$$\underset{x_{i,t}\geq 0, y_{i,t}\in\{0,1\}}{\text{Maximize}} \quad \sum_{t=1}^{T}\sum_{i=1}^{n} u_{i,t}y_{i,t}$$

$$\text{subject to} \quad y_{i,t} \leq \frac{\sum_{s=1}^{t} x_{i,s}}{J_i}, \forall i \in [N], t \in [T], \text{ and} \tag{9.4}$$

$$\sum_{i=1}^{N} x_{i,t} \leq M_t,\ \forall\, t \in [T]. \tag{9.5}$$

If $\mathbf{x}$ is a feasible solution for SYS, then defining $y_{i,t}, \forall i, t$, to be equal to 1 if user i's job is completed by time t and equal to zero otherwise, we observe that $\mathbf{x}, \mathbf{y}$ is a feasible solution to SYS-ILP with the same objective value as that of SYS with $\mathbf{x}$. On the other hand, if $\mathbf{x}, \mathbf{y}$ is a feasible solution for SYS-ILP, then defining $T_i = \min\{t \in [T] : y_{i,t} > 0\}$ for all $i \in [N]$, we get that it forms a feasible solution for SYS with the same objective value. This gives us an equivalence between the problems SYS and SYS-ILP. Problem SYS-ILP can be solved using Mixed Integer Linear Programming (MILP) methods available in computing frameworks such as MATLAB [242] for sufficiently small N and T.

Relaxing SYS-ILP: Although SYS/SYS-ILP are NP-hard in general (see remark 14), they can be solved approximately by relaxing the integer constraints. Let us replace the constraint $y_{i,t} \in \{0, 1\}$ by $0 \leq y_{i,t} \leq 1$. Since $u_{i,t} \geq 0$, the optimal $y^*_{i,t}$ for the relaxed problem is attained when the constraint (9.4) is satisfied with equality for all i, t as long as $y_{i,t} \leq 1$. Hence, we can substitute $y_{i,t} = \sum_{s=1}^{t} x_{i,s}/J_i$ in the objective function and introduce an additional constraint, $\sum_{t=1}^{T} x_{i,t} \leq J_i$, for all i, to ensure that $y_{i,t} \leq 1$. Fractional $y_{i,t}$ represents the fraction of functions completed for user i at time t, and the system is awarded a utility per function depending on the tiers in which these functions are executed unlike depending on the completion time of all functions considered earlier. Letting

$$F_{i,t} := \frac{U_{i,t}}{J_i}$$

denote the utility per unit function, for all i and t, and making appropriate substitutions in SYS-ILP,

we get

SYS-LP

$$\underset{x_{i,t}\geq 0}{\text{Maximize}} \quad \sum_{i=1}^{N}\sum_{t=1}^{T} x_{i,t}F_{i,t}$$

$$\text{subject to} \quad \sum_{t=1}^{T} x_{i,t} \leq J_i,\ \forall\, i \in [N], \tag{9.6}$$

$$\sum_{i=1}^{N} x_{i,t} \leq M_t,\ \forall\, t \in [T]. \tag{9.7}$$

The nice LP form of SYS-LP would later allow us to devise pricing mechanisms for resource allocation, similar to the network resource allocation schemes in [236], [237]. In particular, the dual variable μ_t corresponding to the constraint (9.7) plays the role of an auxiliary price (or a shadow price) for the t-th service tier (more on this in Sec. 9.4).

Let V^* be the optimum sum utility of the system in SYS/SYS-ILP. Let V^R denote the optimum value for the relaxed problem SYS-LP (which is obtained by relaxing the integer constraints on $y_{i,t}$). It is clear that $V^R \geq V^*$, since V^R is obtained by relaxing the integer constraints. Let $\mathbf{x}^R$ be the corresponding resource allocation matrix that achieves the optimum value V^R in SYS-LP. Let $\mathbf{y}^R$ be the corresponding matrix given by

$$y^R_{it} := \sum_{s=1}^{t} \frac{x_{i,s}}{J_i}. \tag{9.8}$$

We define the matrix $\hat{\mathbf{y}} \in \mathbb{R}^{N\times T}$ with entries

$$\hat{y}_{i,t} := \begin{cases} 0, \text{ if } y^R_{i,t} < 1, \\ 1, \text{ if } y^R_{i,t} = 1, \end{cases} \quad \forall i, t. \tag{9.9}$$

We can verify that $\mathbf{x}^R, \hat{\mathbf{y}}$ is a feasible solution for SYS-ILP. Let $\hat{V}$ denote the value of the objective in SYS-ILP at $\mathbf{x}^R, \hat{\mathbf{y}}$. Then, since $\hat{V}$ is one possible solution to SYS-ILP (whose optimal solution is V^*), we have

$$V^R \geq V^* \geq \hat{V}. \tag{9.10}$$

Later, we show that the inequality gap $(V^* - \hat{V})$ is small. Note that this implies that the gap $(V^* - V^R)$ is small. To that end, we first analyze the problem SYS-LP in more detail. (See remark 15 for an alternative way to relax the problem SYS as proposed by [232] and how it compares with our approach.)

9.3 Analyzing SYS-LP

Consider the Lagrangian corresponding to the optimization problem SYS-LP,

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= \sum_{i=1}^{N}\sum_{t=1}^{T} x_{i,t}F_{i,t} + \lambda_i \left(J_i - \sum_{i=1}^{N} x_{i,t} \right) + \mu_t \left(M_t - \sum_{t=1}^{T} x_{i,t} \right) \\ &= \sum_{i=1}^{N}\sum_{t=1}^{T} \left(F_{i,t} - \mu_t - \lambda_i \right) x_{i,t} + \sum_{t=1}^{T} \mu_t M_t + \sum_{i=1}^{N} \lambda_i J_i, \end{aligned} \tag{9.11}$$

where $\mathbf{x} = [x_{i,t}], i \in [N], t \in [T]$, is the primal variable matrix and $\boldsymbol{\lambda} = [\lambda_i], i \in [N]$, and $\boldsymbol{\mu} = [\mu_t], t \in [T]$, are the dual variable vectors corresponding to the constraints in (9.6) and (9.7), respectively. Here, μ is the auxiliary price vector for the T service tiers that comes out of the resource allocation problem. Thus,

$$\begin{aligned} \frac{\partial L}{\partial x_{i,t}} &= F_{i,t} - \mu_t - \lambda_i, \quad \forall i, t; \\ \frac{\partial L}{\partial \lambda_i} &= J_i - \sum_{i=1}^{N} x_{i,t}, \qquad \forall i; \\ \frac{\partial L}{\partial \mu_t} &= M - \sum_{t=1}^{T} x_{i,t}, \qquad \forall t. \end{aligned}$$

Hence, the Karush-Kuhn-Tucker (KKT) conditions [149] imply that

$$\mu_t + \lambda_i \begin{cases} = F_{i,t}, & \text{if } x_{i,t} > 0, \\ \geq F_{i,t}, & \text{if } x_{i,t} = 0, \end{cases} \qquad \forall i, t, \tag{9.12}$$

$$\sum_{i=1}^{N} x_{i,t} \begin{cases} = M_t, & \text{if } \mu_t > 0, \\ \leq M_t, & \text{if } \mu_t = 0, \end{cases} \qquad \forall t, \tag{9.13}$$

$$\sum_{t=1}^{T} x_{i,t} \begin{cases} = J_i, & \text{if } \lambda_i > 0, \\ \leq J_i, & \text{if } \lambda_i = 0, \end{cases} \qquad \forall i. \tag{9.14}$$

We will now show that there exists an optimal solution to the SYS-LP problem with a special structure. Note that when $x_{i,t} > 0$, we have $\mu_t + \lambda_i = F_{i,t}$ from the KKT condition in Eq. (9.12). We know that every time a user i gets non-zero resources allocated in multiple tiers, say t_1 and t_2, we have $\mu_{t_1} + \lambda_i = F_{i,t_1}$ and $\mu_{t_2} + \lambda_i = F_{i,t_2}$. This implies that

$$\mu_{t_1} - \mu_{t_2} = F_{i,t_1} - F_{i,t_2}. \tag{9.15}$$

Hence, every time any user gets partial resources (that is, less than the total job size) allocated in one tier, we get one relation of the form (9.15). If there are more than $T - 1$ instances of such partial

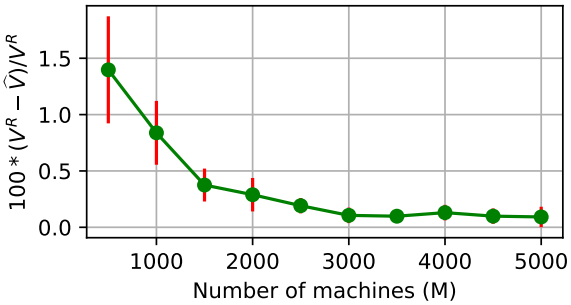


Figure 9.3: Percentage error between V^R (the utility obtained with SYS-LP problem) and $\hat{V}$ (the utility obtained by applying integer constraints on SYS-LP solution).

resource allocation, then we can eliminate the variables $\mu_t, t \in [T]$, to get a non-trivial relation amongst the variables $F_{i,t}$'s. In a generic case, where the $F_{i,t}$'s are any real numbers, it is unlikely that such a non-trivial relation amongst the variables $F_{i,t}$'s is satisfied. Expanding on this idea, we prove the following lemma whose proof is included in Appendix 9.8. (See [243] for a related result in the context of network resource allocation. See also remark 16.)

Lemma 15. *There exists an optimal solution $x_{i,t}$ to SYS-LP such that $|\{(i,t) : 0 < x_{i,t} < J_i\}| \leq T$.*

Recall that as a result of Lemma 14, we know that the optimal resource allocation in SYS is $(N+T)$-sparse. Sparsity in both SYS and SYS-LP solutions suggests that the resource allocation in the two cases is similar. In theorem 12, we formally bound the gap, $V^R - V^*$, between the objectives of SYS and SYS-LP problems.

Theorem 12. *Let V^* and V^R be the optimal objectives of the problem in SYS and SYS-LP, respectively. Then, we have*

$$V^* \geq \left(1 - \frac{T(\max_i J_i)}{\min_t M_t}\right) V^R. \tag{9.16}$$

Thus, the relaxation from SYS-ILP to SYS-LP does not sacrifice much in terms of optimality. This can also be noted from Fig. 9.3, where we plot the mean and standard deviation of the percentage error between V^R and $\hat{V}$, that is, $100 * (V^R - \hat{V})/V^R$ over 20 independent trials for $T = 5$ tiers and $N = 100$ users, where utilities and job sizes are independently and identically distributed (i.i.d.) across users, and $\mathbb{E}[J_i] = 50,\ \forall\, i$. We have taken $M_t (= M)$ constant across tiers. Recall that the plot is an upper bound on the percentage error between V^R and V^*.

Suppose, in addition to the matrix $\mathbf{x}^R$ being $(N+T)$-sparse, we know that the allocation $\mathbf{x}^R$ allocates partial resources to at most one user in any tier t as is typical in a greedy allocation (see Remark 13). Then, in the proof of Theorem 12 (see section 9.8), we have $|S_t| \leq 1$, for all t (where S_t is the set of users that are allocated partial resources in tier t), and we can improve the bound to get,

$$V^* \geq \left(1 - \frac{\max_i J_i}{\min_t M_t}\right) V^R.$$

Premium services are charged more: We now give a toy example to illustrate some of the features of our pricing scheme. Consider a system with $N = 3$ users having utility functions as shown in Fig. 9.1. Let the total job size for each user be 10. Let the cloud provider allocate resources in $T = 3$ tiers corresponding to job completion under $0.1, 10$ and 1000 seconds. Also, let $M_t = 10$ for $t = 1, 2, 3$. Thus, according to the utilities shown in Fig. 9.1, the 3×3 utility matrix $\mathbf{U} = [U_i(t)|\ i \in \{1, 2, 3\}, t \in \{1, 2, 3\}]$ is given by

$$\mathbf{U} = \begin{bmatrix} 3 & 0 & 0 \\ 4 & 2.5 & 1 \\ 2 & 2 & 2 \end{bmatrix}.$$

For both SYS and SYS-LP, the following resource allocation is optimal:

$$x_{i,t} := \begin{cases} 10, \text{ if } i = t, \\ 0, \text{ if } i \neq t, \end{cases} \quad \forall\, i, t \in \{1, 2, 3\}. \tag{9.17}$$

The above allocation makes intuitive sense because it maximizes the sum utility of the system by maximizing the individual utility of the users. Any vector $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3)$ such that $0.15 \leq \mu_1 \leq 0.3$, $0 \leq \mu_2 \leq 0.25$, $0 \leq \mu_3 \leq 0.2$, $\mu_1 - \mu_2 \geq 0.15$ and $\mu_2 \geq \mu_3$ serves as an auxiliary pricing vector for the three tiers corresponding to the above optimal resource allocation. (Observe that if we take $\lambda_1 = 0.3 - \mu_1$, $\lambda_2 = 0.25 - \mu_2$, and $\lambda_3 = 0.2 - \mu_3$, then the KKT conditions (9.12), (9.13), and (9.14) are satisfied.) An example of such an auxiliary pricing vector for the three service tiers is given by $\boldsymbol{\mu} = [0.259, 0.083, 0.048]$ (obtained through the `CVX` optimizer from [149]). Note that in our scheme, the cloud provider sets prices for different service tiers proportional to $\boldsymbol{\mu}$. Thus, user 1 is charged more for getting her job completed in tier 1 as opposed to user 3 who is charged less for being flexible. Note that this is also better than a myopic greedy allocation where user 2 would have been selected first based on her high utility value in the first tier. Our pricing scheme thus takes into account the users' delay-sensitivity and charges each user accordingly. The conditions on $\boldsymbol{\mu}$ show that a flat pricing scheme cannot achieve this result in our toy example setting.

9.4 Pricing Mechanisms to Learn User Utilities

In the previous section, we assumed that the utilities of the users are known to the cloud service provider. This is not true in general. However, through dynamic pricing and users' responses to this pricing, the cloud service provider can converge to the optimal scheduling and pricing. Such pricing mechanisms have been studied in economics for two-sided (supply and demand) markets and are called Walrasian auctions [244], [245]. In the networking literature, Kelly has used similar mechanisms to allocate bandwidth optimally over a network [236], [237].

In this section, we devise dynamic pricing schemes that would help the cloud provider to maximize the utility of the system. To the best of our knowledge, *this is the first work to study dynamic pricing mechanisms for scheduling problems*. This is a promising research direction with potentially high impact on the economics and resource allocation of next-generation computing

systems. Next, we derive an algorithm that sets pricing for cloud resources without assuming that the users' utilities are known.

We first decompose the system problem from the i-th user's perspective and from the cloud provider's perspective. To introduce the user feedback, we define a new variable $m_{i,t}$, which can be thought of as the budget of user i for service interval t. Let q_t be the price set by the cloud service provider at service interval t. Then the amount of resources allocated to the i-th user in service interval t is given by $x_{i,t} = m_{i,t}/q_t$. With this interpretation in mind, we formulate the user problem for the i-th user by taking the terms from the Lagrangian of the system formulation (Eq. (9.11)) that are relevant to the i-th user along with the constraint on the number of functions in her job (Eq. (9.6)).

USER(i)

$$\begin{aligned} \underset{m_{i,t}\geq 0}{\text{Maximize}} \quad & \sum_{t=1}^{T} \frac{m_{i,t}}{q_t} \left(F_{i,t} - q_t\right) \\ \text{subject to} \quad & \sum_{t=1}^{T} \frac{m_{i,t}}{q_t} \leq J_i. \end{aligned} \tag{9.18}$$

The i-th user thus allocates a larger budget $m_{i,t}$ at time t if the user utility per function at time t, $F_{i,t}$, is sufficiently larger than the price q_t set by the cloud service provider. Note that, if $q_t = 0$, then $m_{i,t} = 0$. Because, otherwise, we would have $m_{i,t}/q_t = \infty$, and this would violate constraint (9.18). Given a price vector $\mathbf{q} = [q_t], t \in [T]$, the i-th user solves the USER(i) problem to obtain the budget vector $m_{i,t}, t \in [T]$. The cloud provider then receives budgets from all users $m_{i,t}, t \in [T], i \in [N]$, and solves the CLOUD problem defined as follows:

CLOUD

$$\begin{aligned} \underset{x_{i,t}\geq 0}{\text{Maximize}} \quad & \sum_{t=1}^{T} \sum_{i=1}^{N} m_{i,t} \log x_{i,t} \\ \text{subject to} \quad & \sum_{i=1}^{N} x_{i,t} \leq M_t, \ \forall\, t \in [T]. \end{aligned} \tag{9.19}$$

Theorem 13. *There exist equilibrium matrices* $\mathbf{x} = (x_{i,t}, i \in [N], t \in [T])$ *and* $\mathbf{m} = (m_{i,t}, i \in [N], t \in [T])$*, and an equilibrium price vector* $\mathbf{q} = (q_t, t \in [T])$ *such that*

1. $\mathbf{m}_i = (m_{i,t}, t \in [T])$ *solves USER(i),* $\forall\, i \in [N]$,
2. $\mathbf{x} = (x_{i,t}, i \in [N], t \in [T])$ *solves the CLOUD problem,*
3. $m_{i,t} = x_{i,t} q_t$*, for all* $i \in [N], t \in [T]$,
4. *for any* t*, if* $\sum_i x_{i,t} < M_t$*, then* $q_t = 0$.

Further, if any matrix $\mathbf{x} = (x_{i,t}, i \in [N], t \in [T])$ *that is at equilibrium, i.e. has a corresponding matrix* $\mathbf{m}$ *and a vector* $\mathbf{q}$ *that together satisfy (i), (ii), (iii), and (iv), then* $\mathbf{x}$ *solves the system problem SYS-LP.*

Condition (i) says that the equilibrium budgets $(m_{i,t}, t \in [T])$ form an optimum response by the user i to the equilibrium rates $(q_t, t \in [T])$. Condition (ii) says that the equilibrium allocation matrix $\mathbf{x}$ is the solution to the CLOUD problem with respect to the equilibrium budgets $\mathbf{m}$. Further, the allocations, prices and the budgets are consistent under equilibrium, i.e. $m_{i,t} = x_{i,t}q_t$ as per our interpretation of budgets (from condition (iii)). And finally, (iv) says that the price $q_t = 0$ if that corresponding service interval is not full, i.e. $\sum_i x_{i,t} < M_t$. Besides, in the proof of Theorem 13, we show that if $\boldsymbol{\mu}$ is the dual variable corresponding to an optimal solution $\mathbf{x}$ of SYS-LP, then $\mathbf{q} = \boldsymbol{\mu}$ form an equilibrium price vector.

Price tracking using gradient descent: In many cases, it is easier to work with the dual problem because there are less variables involved. For example, for the CLOUD problem, the primal has NT variables, which can be significantly large in comparison to only T variables in the dual problem (coming from T constraints in the primal). Moreover, working on the dual problem allows the cloud provider to work directly on the price vector $\mathbf{q}$, which is what it shows to the users. We can derive the following dual problem for the CLOUD problem using the Lagrangian $L(\mathbf{x}, \mathbf{q})$ in Eq. (9.31).

$$\underset{q_t \geq 0}{\text{Maximize}} \quad \sum_{t=1}^{T} \log q_t \left(\sum_{i=1}^{N} m_{i,t} \right) - \left(\sum_{t=1}^{T} M_t q_t \right). \tag{9.20}$$

The gradient of the above objective w.r.t. q_t is given by $\left(\frac{\sum_{i=1}^{N} m_{i,t}}{q_t} - M_t \right) \ \forall \ t \in [T]$. We use gradient descent to solve Eq. (9.20) and make the pricing scheme $\mathbf{q}$ track users' budgets $\mathbf{m}$.

Next, based on our results in this section, we derive an algorithm where the cloud provider updates the prices of its resources. Here, we assume that the user utilities are not known to the cloud provider (which is generally the case) but they provide their budget information (represented as the $\mathbf{m}$ matrix in the USER/CLOUD problems) on a semi-regular basis which is determined based on the current price vector (represented as $\mathbf{q}$ in the USER/CLOUD problems) shown by the cloud service provider (see Alg. 19).

An illustration of the process is provided in Fig. 9.4. Further, for $N = 100$ users, $T = 5$ hours and i.i.d. users' utilities and job sizes, we plot the results of Algorithm 19 (with gradient steps $G = 40$ and step-size $\kappa = 10^{-6}$) in Fig. 9.5. After only $\sim$10 iterations (budget/pricing updates), the tracking-based scheme converges to the optimal solution (Fig. 9.5a). Moreover, as shown in Fig. 9.5b, with only slight changes in its pricing scheme, the cloud provider is able to nudge the users to change their budgets to match the optimal solution.

Algorithm 19: Finding the optimal resource allocation and pricing without utility information at the cloud end

Input : Total number of machines M, step-size κ, error tolerance ϵ, gradient steps G, job sizes J_i and utilities $U_i(\cdot)$ that are known only to user i for $i \in [N]$

Initialization: Cloud service provider shows some initial prices $\mathbf{q} = \mathbf{q}_0 \in \mathbb{R}^T$ (say, the all ones vector)

while $\|\mathbf{q} - \mathbf{q}_{prev}\| \geq \epsilon \|\mathbf{q}_{prev}\|$ **do**

  $\mathbf{q}_{prev} = \mathbf{q}$

  Users, for all, $i \in [N]$, solve for budget vector $\mathbf{m}_i \in \mathbb{R}^T$ using the price vector $\mathbf{q}$ in the USER(i) problem

  The cloud service provider receives the budget matrix $\mathbf{m}$ and does the following

  step = 0

  while *step* $\leq G$ **do**

    $q_t = \max\left[q_t + \kappa\left(\frac{\sum_{i=1}^{N} m_{i,t}}{q_t(\tau)} - M_t\right), 0\right] \ \forall\, t \in [T]$

    $step = step + 1$

  end

end

$\mathbf{q}^* = \mathbf{q}, \mathbf{m}^* = \mathbf{m}$

$z_{i,t} = m^*_{i,t}/q^*_t$ for all $i \in [N], t \in [T]$

Use Algorithm 20 to obtain $x^*_{i,t}$, the projection of $z_{i,t}$ to the constraint sets $\sum_i z_{i,t} \leq M_t, \ \forall\, t$ and $z_{i,t} \geq 0 \ \forall\, i, t, k$.

Result: Solution to SYS, that is, the optimal pricing vector $\mathbf{q}^*$ and the optimal resource allocation vector $\mathbf{x}^*$

9.5 A Market Simulation

In this section, we run an experiment for 60 days with $N = 100$ users and $T = 5$ tiers, where each day, users' utilities are changing based on the market trends[3]. Specifically, for the first 30 days, we synthetically generate utilities that follow an upward trend based on the market following by a downward trend for the next 30 days. To simulate this, we generate $u_{it}, \forall\, i, t$, from a uniform distribution in $[5, 10]$ in an i.i.d. fashion. The job size for the i-th user, $J_i \ \forall\, i$, is an i.i.d. integer chosen between 10 to 100 (which remains constant throughout the 60 days of the experiment) and $M_t = 5000$ for all t. To generate an upward market trend, we add 0.5 to $u_{i,t}, \forall\, i, t$, with probability 0.55 and -0.5 with probability 0.45 (the probabilities are flipped to generate a downward trend). [4]

We compare the following three schemes:

- **Optimal pricing**: Here, we assume that the cloud provider is able to solve for optimal

[3] An implementation of the price tracking algorithm (Alg. 19) and the market simulation described in this chapter is available here.

[4] Note that the user problem can have multiple solutions. To ensure convergence to a unique solution with algorithm 19, we add a small quadratic regularizer to both SYS-LP and the user problem.

Algorithm 20: Han's algorithm for projection on the intersection of convex sets [246], [247]

Input : $\mathbf{z} \in \mathbb{R}^{N \times T}$, Constraint sets $C_1, C_2 \subset \mathbb{R}^{N \times T}$, where $C_1 := \{z_{i,t} | \sum_i z_{i,t} \leq M_t, \forall t\}$, and $C_2 := \{z_{i,t} | z_{i,t} \geq 0 \, \forall i, t\}$, error tolerance ϵ

Initialization: Define $\mathbf{z}_1 = \mathbf{z}, \quad \mathbf{z}_2 = \mathbf{z}$.

while $\|\mathbf{z} - \mathbf{z}_{prev}\| \geq \epsilon \|\mathbf{z}_{prev}\|$ **do**

$\mathbf{z}_{prev} = \mathbf{z}$

$(z'_1)_{i,t} = z_{i,t} - \max\left[\frac{\sum_i z_{i,t} - M_t}{N}, 0\right] \; \forall i, t.$

$\mathbf{z}'_2 = \mathbf{z}, \quad \mathbf{z}'_2[\mathbf{z}'_2 < 0] = 0$

$\mathbf{z} = (\mathbf{z}'_1 + \mathbf{z}'_2)/2$

$\mathbf{z}_1 = \mathbf{z} + \mathbf{z}_1 - \mathbf{z}'_1$

$\mathbf{z}_2 = \mathbf{z} + \mathbf{z}_2 - \mathbf{z}'_2$

end

Result: $\mathbf{z}$: Projection of the input matrix to sets C_1 and C_2

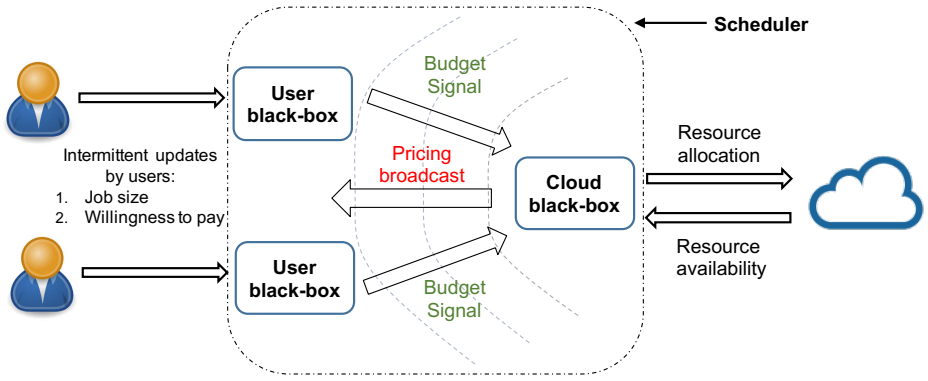
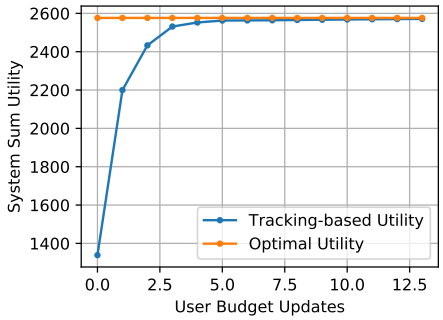


Figure 9.4: Decomposition into user and cloud problems. Here, we visualize a user black-box for each user i that solves the USER(i) problem using the prices shown by the cloud black-box, the unscheduled functions in the job of user i and her willingness to pay (intermittently updated). The cloud black-box runs Algorithm 19 to update the prices using the budget signals from the user black-boxes and the capacity constraints from the cloud.

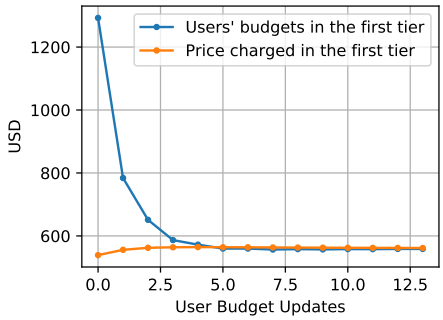
resource allocation and maximize system utility by utilizing the knowledge of users' utilities at each day.

- **Tracking-based pricing**: In this case, the users' utilities are not known, and the cloud provider tracks users' utilities based on their budget signals (as described in Algorithm 19). The cloud provider is assumed to update the prices everyday. *Users send budget signals only once per day.* These budget signals depend on the user's utility function on that day and the prices published by the cloud on that day.

- **First-come-first-serve** (also known as first-in-first-out): Here, the pricing of the resources



(a) After less than 10 budget updates, the system utility obtained through Algorithm 19 approaches the optimal utility.



(b) After less than 10 budget updates, the prices and users' budgets reach an equilibrium.

Figure 9.5: Finding the SYS-LP solution through price tracking (Algorithm 19). The algorithm converges after $\sim$10 iterations.

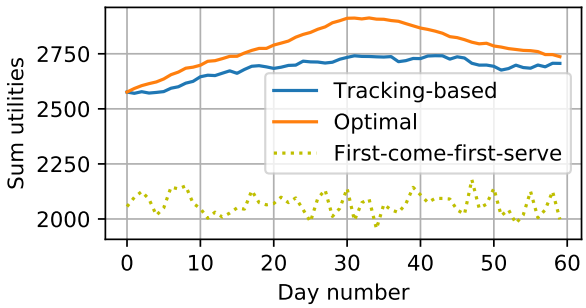
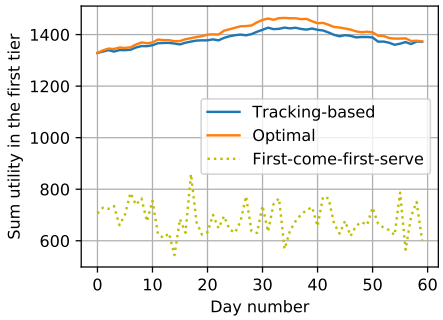


Figure 9.6: Sum utilities for three resource allocation schemes

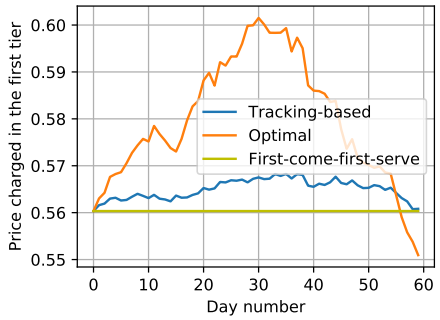
remains constant (as currently employed by most commercial service providers). It is the optimum prices determined by the utilities on day 1 and does not capture the mood of the market. Each day, the users are allocated resources on a first-come-first-serve (which is assumed to be random in order) or a lottery basis.

In Fig. 9.6, we plot the sum utility of the system for all the three schemes across 60 days. Note that the tracking-based scheme accurately captures the trends of the market and is always within 8% of the optimal utility, and this happens with only limited feedback from users (where they send budget signals only once per day). The first-come-first-serve scheme is clearly suboptimal with a deviation of as much as 38% from the optimal utility.

In Fig. 9.7, we plot the utility and price charged across all users only in the first tier. Again, the tracking-based scheme closely follows the optimal utility and lies within 3% of it (Fig. 9.7a). Furthermore, an important advantage of the tracking-based scheme is that it does not drastically change the prices throughout the sixty days (the maximum change is $< 2\%$, see Fig. 9.7b). This is unlike the optimal pricing scheme, where the change is 10%. From a cloud service provider

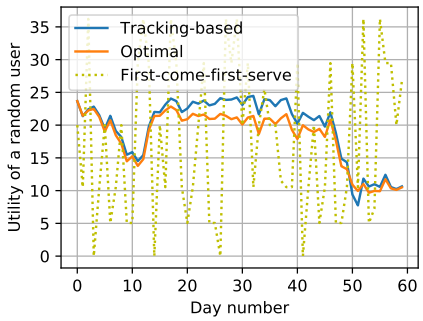


(a) Sum utility in the first tier across 60 days

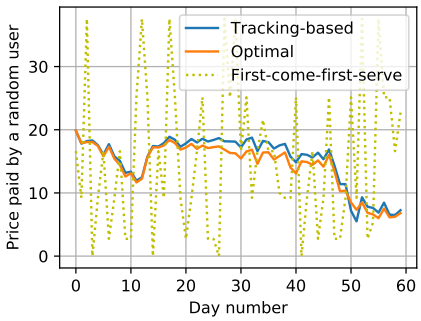


(b) Corresponding price charged by the cloud provider

Figure 9.7: Utility and price charged in the first tier



(a) Utility of a single user over 60 days



(b) Corresponding price charged

Figure 9.8: Utility and price paid by a randomly chosen user

point of view, the optimal scheme is not ideal since the pricing fluctuates heavily and may alienate users who expect some consistency in the pricing scheme. First-come-first-serve keeps a constant pricing scheme, but it reduces the optimal utility by a factor of 2. Hence, the tracking-based scheme represents an optimal trade-off between utility and fluctuation in pricing (and this trade-off can be controlled by the number of gradient steps in Algorithm 19). Moreover, the tracking-based scheme does not require the utility information from users, alleviating potential privacy concerns.

In Fig. 9.8, we pick a random user and plot her obtained utility and price paid throughout the sixty days. Again, the tracking-based utility is extremely close to the optimal utility of the user. Furthermore, even though the pricing obtained through the tracking-based scheme does not fluctuate much, we see that the revenue obtained from the user is very close to the optimal revenue. This is because by slightly changing the prices according to the mood of the market, the cloud provider is able to nudge the users to update their budget to go close to the optimal budgets. Also note that the price charged to the user on each day (Fig. 9.8b) is less than the utility/willingness-to-pay of that user on that particular day (Fig. 9.8a).

9.6 Future work

Our analysis and simulations show that there are several advantages of using dynamic multi-tier pricing over basic pricing. This is only the first step towards designing practical game-theoretic resource allocation schemes for the cloud. Below, we describe some limitations of current work, and consequently, potential directions for future research, which are of interest to both industry and academia.

Utility-based pricing for serverful systems: The schemes developed in this paper can be extended to the serverful case provided the assumption that one function requires one machine and unit time holds. This assumption is more restrictive for the serverful case since the jobs can be large. However, in such scenarios, a large job can be broken into several smaller jobs, each of which requires unit time and machine. But this also introduces dependencies between jobs of a user which need to be taken into account in problem formulation. We explain such job dependencies in more detail next.

Job dependencies: Often, certain jobs are recurrent and require execution of some other specific jobs for their execution. These dependencies are generally represented as a graph. An important future direction is to design improved schedulers that take into account such job dependencies. Ideally, jobs which have low dependencies or that have more reliable request-for-execution times should get a discount in their pricing because they allow the scheduler to plan more efficiently.

Wholesale discount: Current pricing schemes lack a wholesale discount for customers who are requesting large number of jobs with low variability in job sizes. Scheduling such jobs is cheaper for the provider since it can rent entire clusters of machines together, resulting in better resource utilization due to efficient bin packing.

Flexible SLAs with probabilistic guarantees: Cloud service providers define strict Service Level Agreements (SLAs) for their services, e.g. in AWS Lambda, users are credited 10% of their incurred charges if the error rate of Lambda functions goes beyond 0.05%[5]. Note that providing such strict and premium SLAs require high-level maintenance of the cloud, the costs of which are indirectly borne by all the users. This may be unfair since some users could be equipped to tolerate job failures [36], [147] and different users could have varying degrees of fault-tolerance. An interesting future direction is to design pricing schemes that conform with each user's preferences using behavioral preference models from decision theory (see, for example, [248]).

Heterogeneity in machines and jobs: Currently, we have assumed that all the serverless machines/jobs have the same specifications in terms of their execution time, memory capacity, etc. However, there is often heterogeneity both in the types of machines and jobs for such systems [229], which can be taken into account in the problem formulation.

Revenue optimal schemes: In this paper we have focused on optimizing social welfare. An alternative criteria would be to optimize the revenue generated by the cloud provider.

Stability analysis: Our algorithm is provably convergent to the optimal allocation and pricing when the conditions are static, and seems to track the optimum when the network conditions vary

[5]For details, see https://aws.amazon.com/lambda/sla/historical/

slowly. The impact of limited feedback on the ability to track the optimum will be a topic of further studies.

9.7 Related Work

Notice that the two key features of our model are capturing the users' delay-sensitivity and taking a utility-agnostic approach. In the next couple of paragraphs, we recall some of related works in the literature with respect to these two features.

In relation to the first feature, suppose we ignore the market setting, then our problem is closely related to the problem of scheduling jobs in real-time systems. This problem has been extensively studied in the literature. (See, for example, [249]–[252].) The notion of utility as a function of completion times has played a key role in many of these works. Varied algorithms with supporting simulations have been been proposed in these works. Building on these works, many have proposed resource allocation in the cloud that is based on maximizing the utility of the system. For example, in [253] and [254], the authors propose utility-function based approaches for resource allocation in *autonomic computing systems*. In [255], the authors use utility functions to allocate resources dynamically and save energy by consolidating load. In [256], the authors propose schemes to dynamically allocate computing resources to virtual machines (such as virtual operating systems) while minimizing operating costs and satisfying QoS constraints. This is done by expressing these two goals as a two-tier utility function. Another example is [257], where the authors find the optimal number and size of virtual machines to allocate CPU resources to applications via an automatic resource controller. This is done by using a constraint programming approach to maximize the utility accrued. Thus, using system utility maximization as the overall objective for designing resource allocation theme has been a recurring theme in the literature. Our objective in this paper shares this feature with these works.

Pricing in cloud computing is closely related to the second feature, although to the best of our knowledge this idea has not been fully exploited in the literature. When cloud computing was first introduced in [258], the authors claimed that the success of the cloud can only be obtained by developing adequate pricing schemes. Since then a variety of pricing schemes for the cloud have been proposed. Here, we give a small sample of the several works that propose and analyze dynamic pricing schemes for various services in the cloud. In [259], the authors propose an iterative pricing algorithm that uses the historical pricing of resources and determines the final price based on availability of resources for the next round. In [260], the authors analyze four dynamic pricing schemes and develope an agent-based simulation of a software market. In [261], the authors propose a federated version of dynamic pricing where the computing resources are being shared by multiple cloud service providers. Finally, Amazon offers spot pricing, which is another form of dynamic pricing where the resources are priced at a lower rate than fixed pricing but with less guarantee of availability [106]. For further examples of pricing schemes, see [105].

9.8 Proofs

Proof of Lemma 14. Let $\mathbf{x}$ be any optimal resource allocation to SYS and let T_i the corresponding end times as defined in (9.1). Without loss of generality, let the following ordering hold:

$$T_1 \leq T_2 \leq \cdots \leq T_N. \tag{9.21}$$

As defined above, let $\tilde{\mathbf{x}}$ be the greedy allocation corresponding to the ordering $1 \leftarrow 2 \leftarrow 3 \leftarrow \cdots \leftarrow N$. Let $\tilde{T}_i$ be the end times corresponding to the allocation $\tilde{\mathbf{x}}$. By construction of the greedy allocation, we have

$$\tilde{T}_i = \min \left\{ t \in [T] : \sum_{s=1}^{t} M_t \geq \sum_{j=1}^{i} J_j \right\}, \text{ for all } i \in [N].$$

On the other hand, because of the ordering (9.21), we have

$$T_i \geq \min \left\{ t \in [T] : \sum_{s=1}^{t} M_t \geq \sum_{j=1}^{i} J_j \right\}, \text{ for all } i \in [N].$$

Thus, $\tilde{T}_i \leq T_i$, for all $i \in [N]$. Hence $\sum_i U_{i,\tilde{T}_i} \geq \sum_i U_{i,T_i}$ and, thus, $\tilde{\mathbf{x}}$ is also an optimal allocation. Since $\tilde{\mathbf{x}}$ is a non-preemptive scheduling, we have the statement in the lemma. □

Proof of Lemma 15. We will first extend the SYS-LP problem by adding a dummy service tier $T+1$ with unlimited capacity $M_{T+1} = \infty$. Let $U_{i,(T+1)} = 0$ for all users $i \in [N]$. Let us call this problem SYS-LP-EXT. For any feasible solution $\mathbf{x}$ to SYS-LP, we can construct a feasible solution $\tilde{\mathbf{x}}$ to SYS-LP-EXT by executing all the remaining functions in tier $T+1$, i.e. for all $i \in [N], t \in [T+1]$, let

$$\tilde{x}_{i,t} := \begin{cases} x_{i,t}, & \text{if } i \in [N], t \in [T], \\ J_i - \sum_{t \in [T]} x_{i,t}, & \text{if } i \in [N], t = T+1. \end{cases}$$

Similarly, we can construct a solution $\mathbf{x}$ to SYS-LP corresponding to any solution $\tilde{\mathbf{x}}$ to SYS-LP-EXT by restricting it to $i \in [N]$ and $t \in [T]$. Since $U_{i,(T+1)} = 0$ for all users $i \in [N]$, we have that the objective values for SYS-LP and SYS-LP-EXT match for corresponding feasible solutions. We will now show that there exists an optimal solution $\tilde{x}$ to SYS-LP-EXT such that $0 < \tilde{x}_{i,t} < J_i$ for at most T elements. Taking the corresponding solution for SYS-LP, we will get the optimal solution with the desired properties for SYS-LP.

Let $\tilde{\mathbf{x}}$ be an optimal solution to SYS-LP-EXT. Without loss of generality, let us assume that all users get their jobs completed by the end of tier $T+1$, i.e. $\sum_t \tilde{x}_{i,t} = J_i$, for all i. We have this because tier $T+1$ is assumed to have infinite capacity. Consider the KKT conditions to problem SYS-LP-EXT similar to (9.12), (9.13), and (9.14), with dual variables $\tilde{\mu}_t, t \in [T+1], \tilde{\lambda}_i, i \in [N]$. Since $M_{i,(T+1)} = \infty$, we have $\tilde{\mu}_{T+1} = 0$. If $\tilde{x}_{i,t} > 0$, then $\tilde{\mu}_t + \tilde{\lambda}_i = F_{i,t}$. Suppose user i is allocated partial resource in some tier $t_1 \in [T]$, i.e $0 < \tilde{x}_{i,t_1} < J_i$. Then there exists a another tier $t_2 \in [T+1]$ such that $0 < \tilde{x}_{i,t_2} < J_i$. We will then have

$$\tilde{\mu}_{t_1} - \tilde{\mu}_{t_2} = F_{i,t_1} - F_{i,t_2}.$$

Corresponding to any instance (i, t_1, t_2) consider the equation above. If we have any $(T+1)$ such distinct instances, then we can eliminate the μ_t variables and get a non-trivial equality relationship between the variables $F_{i,t}$. For each collection of $(T+1)$ distinct instances (i, t_1, t_2), we get a non-trivial equality relationship between the variables $F_{i,t}$. Suppose for the moment that the variables $F_{i,t}$ are such that they do not satisfy any of the equality relationships obtained from $(T+1)$ distinct instances (i, t_1, t_2). Then, we get that $0 < \tilde{x}_{i,t} < J_{i,t}$ for at most T elements. This would give us the required result. In the rest of the proof, we will extend this argument to any variables $F_{i,t}$.

Consider a neighborhood $\mathcal{N}$ in the nonnegative orthant of the $N \times (T+1)$-dimensional Euclidean space around the matrix $(F_{i,t})_{i\in[N],t\in[T+1]}$. Consider the set of points $\mathcal{P}$ in this neighborhood such that they do not satisfy any of the equality relationships obtained from $(T+1)$ distinct instances (i, t_1, t_2). We claim that the set $\mathcal{P}$ is dense in the set set $\mathcal{N}$. To see this, note that the set of points in $\mathcal{N}$ that satisfy any given non-trivial equality relationship is zero. Since there are finitely many such equality relationships that we need to consider, we get that the set of points in $\mathcal{N}$ that satisfy any of these non-trivial equality relationship is zero. Since $\mathcal{P}$ is the complement of this set, it is dense in $\mathcal{N}$. As a result, we get that there is a sequence of points $(F^l_{i,t})_{i\in[N],t\in[T+1]}$, $l \geq 1$, belonging to the set $\mathcal{P}$ and converging to $(F_{i,t})_{i\in[N],t\in[T+1]}$. Since the problem SYS-LP-EXT has a continuous objective function and continuous contraint functions, we get that there exists a sequence of solutions $\tilde{\mathbf{x}}^l, l \geq 1$ such that $\tilde{\mathbf{x}}^l$ is an optimal solution to the SYS-LP-EXT problem with variables $(F_{i,t})_{i\in[N],t\in[T+1]}$ replaced by $(F^l_{i,t})_{i\in[N],t\in[T+1]}$, and $\tilde{\mathbf{x}}^l$ is convergent. Let it converge to $\tilde{\mathbf{x}}'$. We note that $\tilde{\mathbf{x}}'$ is an optimal solution to SYS-LP-EXT (see [262]). Then we get that $\tilde{\mathbf{x}}^l$ has at most T elements such that $0 < \tilde{x}^l_{i,t} < J_i$. This implies that $\tilde{\mathbf{x}}'$ has at most T elements such that $0 < \tilde{x}'_{i,t} < J_i$. This completes the proof. □

Proof of Theorem 12. Suppose the resource allocation matrix $\mathbf{x}^R \in \mathbb{R}^{N\times T}$ is such that there are at most T elements such that $0 < x_{i,t} < J_i$. We know that such an optimal solution exists from Lemma 15. In the optimal resource allocation $\mathbf{x}^R$ for SYS-LP, say a user i is getting non-zero resources in slots m and n (and say $m < n$), i.e. $x_{i,m} > 0, x_{i,n} > 0$ and a user j which is getting resources in slot m, i.e. $x_{j,m} > 0$. Then, by the optimality of $\mathbf{x}^R$, we have

$$F_{j,m} - F_{j,n} \geq F_{i,m} - F_{i,n}. \tag{9.22}$$

We can easily prove the above by redistributing $\epsilon(> 0)$ fraction of the job from user j in slot m to user i in slot m (and vice versa in slot n). But we know that this can only decrease the objective function in SYS-LP, that is

$$(F_{i,m} - F_{i,n})\epsilon - (F_{j,m} - F_{j,n})\epsilon \leq 0,$$

which proves Eq. (9.22).

Now, we are ready to prove the theorem. From Eq. (9.10), we get that

$$\frac{V^R - V^*}{V^R} \leq \frac{V^R - \hat{V}}{V^R}$$

where $\hat{V}$ is the objective of SYS-LP obtained by projecting solution of SYS-LP to integer constraints as described in Eq. (9.9). Now, our aim is to upper bound the RHS above to get an upper bound on the gap. To that end, we bound the numerator and denominator at each time slot $t \in [T]$. Let V_t^R and $\hat{V}_t$ be the corresponding utilities obtained only at tier $t \in [T]$, i.e.

$$V_t^R := \sum_{i=1}^{N} u_{i,t} y_{i,t}^R \quad \text{and} \quad \hat{V}_t := \sum_{i=1}^{N} u_{i,t} \hat{y}_{i,t},$$

where $y_{i,t}^R$ and $\hat{y}_{i,t}, \forall i, t$, are as defined in (9.8) and (9.9), respectively. Let T_i^R be the time at which the job of user i are getting finished according to the resource allocation $\mathbf{x}^R$, that is,

$$T_i^R := \min\{t \in [T] : \sum_{s=1}^{t} x_{i,s}^R \geq J_i\}.$$

Let $T_i^R = T + 1$, if $\sum_{t=1}^{T} x_{i,s}^R < J_i$. Thus, for tier t, we have

$$\frac{V_t^R - \hat{V}_t}{V_t^R} = \frac{\sum_{i \in S_t} (F_{i,t} - F_{i,T_i^R}) x_{i,t}^R}{\sum_{j=1}^{N} F_{jt} x_{j,t}^R}, \tag{9.23}$$

where S_t is the set of users for which $0 < x_{i,t}^R < J_i$. Formally, $S_t := \{i \in [N], 0 < x_{i,t}^R < J_i\}$. We can further write

$$\frac{V_t^R - \hat{V}_t}{V_t^R} \leq \sum_{i \in S_t} \frac{(F_{i,t} - F_{i,T_i^R}) x_{i,t}^R}{\sum_{j=1}^{N} (F_{j,t} - F_{j,T_i^R}) x_{j,t}^R} \leq \sum_{i \in S_t} \frac{(F_{i,t} - F_{i,T_i^R}) x_{i,t}^R}{(F_{i,t} - F_{i,T_i^R}) \sum_{j=1}^{N} x_{j,t}^R}, \tag{9.24}$$

where the last inequality uses Eq. (9.22). Now, since at time t, user i is getting fractional resources, it implies that the system is operating at full capacity, that is $\sum_{j=1}^{N} x_{j,t} = M_t$. Hence, we get

$$\frac{V_t^R - \hat{V}_t}{V_t^R} \leq \sum_{i \in S} \frac{x_{i,t}^R}{M_t} \leq \sum_{i \in S} \frac{\max_i(J_i)}{M_t}, \tag{9.25}$$

where the last inequality uses the fact that $x_{i,t}^R \leq \max_i(J_i)$. Also, since there are at most T instances where users are getting partial resources, $|S_t| \leq T$, and, we get

$$\frac{V_t^R - \hat{V}_t}{V_t^R} \leq \frac{T \max_i(J_i)}{M_t} \leq \frac{T \max_i(J_i)}{\min_t M_t}. \tag{9.26}$$

Thus,

$$V^R - V^* \leq V^R - \hat{V} = \sum_{t=1}^{T} (V_t^R - \hat{V}_t) \leq \frac{T \max_i(J_i)}{\min_t M_t} \sum_{t=1}^{T} V_t^R = \frac{T \max_i(J_i)}{\min_t M_t} V^R, \tag{9.27}$$

Rearranging, we get

$$V^* \geq \left(1 - \frac{T(\max_i J_i)}{\min_t M_t}\right) V^R,$$

which proves the desired result. □

Proof of Theorem 13. Let $\mathbf{x}$ be an optimal solution to SYS-LP and let $\boldsymbol{\mu}$ and λ be the dual variables corresponding to this solution. We know that these satisfy the KKT conditions (9.12), (9.13), and (9.14). Let $\mathbf{q} = \boldsymbol{\mu}$ and $m_{i,t} = x_{i,t}\mu_{i,t}$ for all i, t.

We will now show that $\mathbf{m}_i$ solves USER(i) for this $\mathbf{q}$. Observe that $m_{i,t} = 0$ if $q_t = 0$ because of the way we have defined $\mathbf{m}$ here. Thus, it is enough to look at the tiers for which $q_t \neq 0$. Hence, without loss of generality, we will assume that $q_t \neq 0$ for all t. Consider the Lagrangian for the user problem USER(i),

$$L(\mathbf{m}_i, p_i) = \sum_{t=1}^{T} \frac{m_{i,t}}{q_t} (F_{i,t} - q_t) + p_i \left(J_i - \frac{m_{i,t}}{q_t} \right), \forall i \in [N], \tag{9.28}$$

where p_i is the dual variable corresponding to the job size constraint (9.18) in the user problem. The KKT conditions can, thus, be written as

$$\mu_t + p_i \begin{cases} = F_{i,t}, & \text{if } m_{i,t} > 0 \\ \geq F_{i,t}, & \text{if } m_{i,t} = 0, \end{cases} \qquad \forall i, t, \tag{9.29}$$

$$\sum_{t=1}^{T} \frac{m_{i,t}}{q_t} \begin{cases} = J_i, & \text{if } p_i > 0 \\ \leq J_i, & \text{if } p_i = 0, \end{cases} \qquad \forall t. \tag{9.30}$$

Taking $p_i = \lambda_i$, we get that these KKT conditions are satisfied. Thus, $\mathbf{m}_i$ is an optimal solution to USER(i) with $\mathbf{q} = \boldsymbol{\mu}$.

Now we will show that $\mathbf{m}$ is an optimal solution for the CLOUD problem. The Lagrangian for the CLOUD problem is given by

$$L(\mathbf{m}, \tilde{\mathbf{q}}) = \sum_{t=1}^{T} \sum_{i=1}^{N} m_{i,t} \log x_{i,t} + \sum_{t=1}^{T} q_t \left(M_t - \sum_{i=1}^{N} x_{i,t} \right), \tag{9.31}$$

where $\tilde{\mathbf{q}} = (\tilde{q}_t, t \in [N])$ is the dual variable corresponding to the constraint (9.19) in the CLOUD problem. Let $\tilde{q} = \mu$. If $m_{i,t} > 0$, then $x_{i,t} > 0$, and differentiating the Lagrangian with respect to $x_{i,t}$ we get

$$\frac{\partial L(\mathbf{m}, \mathbf{q})}{\partial x_{i,t}} = \frac{m_{i,t}}{x_{i,t}} - q_t = 0.$$

If $m_{i,t} = 0$, then $\frac{\partial L(\mathbf{m}, \tilde{\mathbf{q}})}{\partial x_{i,t}} \leq 0$, since $q_t \geq 0$. Further, from (9.13), we have

$$\sum_{i=1}^{N} x_{it} \begin{cases} = M_t, & \text{if } q_t > 0 \\ \leq M_t, & \text{if } q_t = 0, \end{cases} \qquad \forall t. \tag{9.32}$$

Thus, $\mathbf{x}$ and $\mathbf{q}$ satisfy the KKT conditions for the CLOUD problem. Hence $\mathbf{x}$ is an optimal solution to CLOUD with $\mathbf{m}$.

Thus we have showed statements (i) and (ii) in Theorem 13. Statement (iii) follows from construction and statement (iv) follows from (9.13). We now prove the later assertion, namely, if

we have an equilibrium solution $\mathbf{x}, \mathbf{m}, \mathbf{q}$ that satisfy (i), (ii), (iii), and (iv), then $\mathbf{x}$ solves the system problem SYS-LP. To see this, let $\mathbf{x}, \mathbf{m}, \mathbf{q}$ be such an equilibrium solution. Take $\boldsymbol{\mu} = \mathbf{q}$. Since $\mathbf{m}_i$ is an optimal solution to USER(i) with $\mathbf{q}$, there exists dual a variable p_i corresponding to the constraint (9.18). Take $\lambda_i = p_i$ for all i. It is easy to check that $\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}$ satisfy the KKT consitions (9.12), (9.13), and (9.14), and hence, form an optimal solution to SYS-LP. This completes the proof of the theorem. □

9.9 Remarks

Remark 13. Corresponding to the greedy allocation with a given ordering, when the cloud provider moves from tier t to $t+1$, there can be at most one user with a partially satisfied job, for all $t \in [T]$. Hence, there are at most T instances such that the users jobs are partially allocated, that is, $|\{(i,t) : 0 < x_{i,t} < J_i\}| \leq T$. Further, there are at most N entries in the matrix $\mathbf{x}$ such that $x_{i,t} = J_i$. Thus, the allocation matrix $(x_{i,t}), i \in [N], t \in [T]$ can have at most $(N+T)$ non-zero enties, i.e. it is $(N+T)$–sparse.

Remark 14. We note that the problems SYS/SYS-ILP are NP-hard in general. In [263], the authors consider a job scheduling problem for N jobs, with execution times $\{J_1, \ldots, J_N\}$, to be executed on a single machine, where each job has a corresponding due date d_i. The utility function for each job is assumed to decrease by its *tardiness* defined as the delay in the completion of job i from its due date d_i. Namely, if T_i is the completion time of job i, then let $D_i(T_i) := \max\{0, T_i - d_i\}$ be the tardiness. The authors show that the problem of finding a schedule that minimizes the total tardiness is NP-hard in general (see [263]–[265]). Further, this is proved for the case when the parameters J_i and d_i, for all i, take integer values. We note that this is a special case of our problem. To see this, recall that the parameters in our problem are $N, T, (U_{i,t})_{i\in[N],t\in[T]}, (M_t)_{t\in[T]}$. Let N be same as the number of different jobs in [263]. Let $T = \sum_{i=1}^{N} J_i$. Think of $\tau_t = t$, for $t \in [T]$. Thus, each service tier has a unit time interval. Let the utility functions be $U_i(\tau) = D_i(T+1) - D_i(\tau)$. Thus, if T_i is the completion tier for player i, then her utility is given by $U_{i,T_i} = D_i(T+1) - D_i(T_i)$, for all i. Let the resource capacities be $M_t = 1$ for all $t \in [T]$. With this setting, we now observe that the problem of maximizing social welfare is equivalent to minimizing the total tardiness. Given the result in [263], we get that our problem is NP-hard, too, in general.

Remark 15. In [232], the authors arrive at a problem formulation that is similar to SYS. They note that the condition (9.3) is not differentialble, thus making it hard to solve the problem SYS. They propose to approximate this condition by the following equation

$$\hat{T}_i := \frac{1}{\beta} \log \left(\frac{1}{J_i} \sum_{t=1}^{T} e^{\beta t} x_{i,t} \right). \tag{9.33}$$

Note that as $\beta \to \infty$,

$$\hat{T}_i \to \max\{t \in [T] : x_{i,t} > 0\}.$$

If we assume that user i's job is completed, i.e. $\sum_{t=1}^{T} x_{i,t} = J_i$, then $\hat{T}_i \to T_i$ as $\beta \to \infty$. The authors replace U_{i,T_i} with $U_i(\hat{T}_i)$ and use Taylor series first order approximation to further simplify

the optimization problem, which can then be solved analytically. Our approach here is very different from theirs. In the next section, we make an important observation regarding the structure of the optimal solution to SYS-LP (see lemma 15). This observation not only allows us to show that the gap between SYS-LP and SYS is small, but also explains why it is enough to consider our direct relaxation instead of the one like (9.33). Besides, our direct relaxation is useful in decomposing the problem into a cloud problem and several user problems–one for each user– giving rise to a natural dynamic pricing mechanism (see section 9.4). Moreover, in a forthcoming paper, we extend our problem formulation to account for uncertainties in the delay times. Our direct relaxation is particularly useful in this extension.

Remark 16. Consider a scenario where the P_{it}'s are sampled independently from non-atomic probability distributions over some finite intervals $[\overline{P_{i,t}}, \underline{P_{i,t}}]$ (where $\overline{P_{i,t}} < \underline{P_{i,t}}$), for all i, t. (A probability distribution over the real numbers is non-atomic if the probability of any single real number occuring is zero.) In such a scenario, we observe that the probability of the event that a non-trivial relationship amongst the variables $P_{i,t}$'s holds is zero. Thus, for any choice of $T+1$ instances (i, t_1, t_2) that satisfy (9.15), we get a non-trivial relationship amongst the variables $P_{i,t}$'s and the probability of this happening is zero. Since there are finitely many choices for such $T+1$ instances (i, t_1, t_2), we get that the probability of there being more than T partial allocations is zero. Thus, in a generic case in the sense above, for any optimal solution $x_{i,t}$ to SYS-LP, there can be at most T instances where the users are allocated partial resources, i.e. $0 < x_{i,t} < J_i$. Moreover, there are at most N instances where the users get their jobs completed, i.e. $x_{i,t} = J_i$. And hence, $x_{i,t}, i \in [N], t \in [T]$ is $(N+T)$-sparse (cf. Remark 13).

Chapter 10

Cumulative Prospect Theory Perspective

In this chapter, we propose a model for lottery allocations in serverless computing which uses cumulative prospect theory to capture the behavior of humans towards risk and uncertainty.

10.1 Introduction

Cloud computing has given rise to a growing trend of customers opting for computing resources as a service instead of maintaining such systems on their own, allowing them to relieve the burden of back office operations. When consumers adopt such service-oriented architectures, the two most important aspects that they care about are the quality and the reliability of the service [230]. Different customers have different service requirements. With the growing customer demand for cloud computing resources, we must allocate the limited resources optimally amongst the customers based on their needs. In [266] (also Chapter 9), we focused on the quality of service aspect by taking into account the *delay-sensitivity* characteristics of the customers while allocating resources. We saw how to allocate resources that would prioritize the customers based on their delay-sensitivity requirements and charge them appropriately. In this paper, we will extend it further to account for the different reliability requirements of the customers.

Let us first briefly recall the notion of delay-sensitivity and the key contributions from [266]. The delay for a user's job is defined as the time lapsed since the user submits a request for her job (or her job gets triggered due to some event) until the job is completed. Given that it is a key aspect of the quality of service for the user, her utility from the job is defined to be a function of the delay in her job completion. Each user is assumed to have a utility function that maps the delay in her job completion to her corresponding utility from this job. The delay-sensitivity of the user refers to this utility function.

In [266], the authors set the goal of maximizing the total social welfare, i.e. the sum of the utilities received by all the users. This gives rise to a scheduler problem, as shown in figure 10.1, where it takes as its input the job sizes and the utility functions for each user, the capcity contraints or machine availability information from the cloud, and outputs a job allocation schedule.

The scheduler is assumed to operate periodically (say every 0.1 seconds) and schedule the jobs

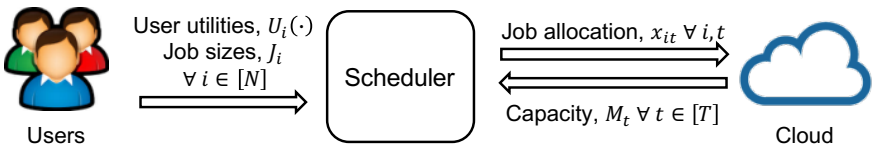


Figure 10.1: A block diagram schematic for the system problem

that are currently in its queue to different service tiers based on their execution times. The different service tiers describe the time of execution of a job and assuming that the job is executed reliably, it marks the completion time of the job. This leads to a multi-tier scheduling problem that must be solved periodically for each scheduler operation.

In [266], the authors formulate this as an optimization problem called the system problem and analyse it further, including certain relaxations of this problem and the gap between the relaxation and the original problem. Besides, they show that the relaxed system problem can be decomposed in a cloud problem and several user problems–one for each user–that together solve the relaxed system problem. This is similar to the result in [236] on the decompostion of network resource allocation problem that underlies the window-based bandwidth allocation method in TCP protocol [238]–[240]. In the cloud setting, this decomposition gives rise to a dynamic multi-tier pricing scheme that incentizes users to bid optimally for resources that are tailored to their delay-sensitivity requirements.

Although such a market-based pricing scheme allows us to differentiate between the users and allocate the resources to customers who "want" them the most, or rather who are ready to pay the most, it has a potential problem. This problem is accentuated when the demand for resources exceeds the supply by a lot. It drives the prices high and discourages some of the smaller customers from participating in such markets. Indeed, the customers come in all sizes. In particular, small business owners particularly rely on cloud computing services as it allows them to off-load the architecture and back-end duties that are critical to maintaining the computing resources and focus on their core business processes [108].

Several service providers use lottery-based allocations to decide which agents to serve next. For example, First-in-First-out (also known as First-Come-First-Serve) is a popular way of allocating warm queues in serverless computing [109]. As a result, all the customers who are willing to pay the minimum price set by the service provider get a shot at receiving the service. Such lottery-based provisions give rise to uncertainties in the execution of the jobs and their delay times. Just as we distinguished between different customers with varying delay-sensitivity, another important criteria for service requirement is the uncertainty in execution of jobs, and different agents have different preferences towards these uncertainties. Our goal in this paper is to capture these varying preferences towards uncertainties and allocate resources optimally via lotteries.

Suppose a user is alloacted a lottery over her delays where the user's job is completed at $\tau = 1s$ with a 20% chance, at $\tau = 5s$ with a 20% chance, and at $\tau = 100s$ with a 60% chance. A common technique to compute the utility or the "happiness" of a user for such a delay-lottery is to evaluate the expected utility for it. However, it has been observed through several empirical studies that

human beings often do not exhibit behavior that conforms with this evaluation technique.

Consider a hypothetical situation where a user's willingness-to-pay is $\$1$ to get her job completed in 1 sec. As in [266], here we will consider the notion of willingness-to-pay in lieu of utilities. Due to an exceeding demand, suppose the market price is $\$2$ for job completion within 1 sec. Thus, the user is unwilling to purchase it at the market price. However, it is possible that this user is ready to pay $\$0.5$ to get a 20% chance of having her job completed in 1 sec. Note that she is ready to pay more than the chance adjusted market price, which would be $0.2 \times \$2 = \$0.4$ in this case. Here, rather than losing out completely on the opportunity to get her job completed within $1s$, she might be ready to make a payment of $\$0.5$, still within her budget of $\$1$ to get a chance of completing the job within $1s$. Note that this behavior is different from the expected utility predictions which say that the user's willingness-to-pay for a 20% chance of getting the job done within $1s$ must be $0.2 \times \$1 = \$0.2$. Customers are known to show such varying degrees of sensitivities to probabilities.

The correct model to consider to account for this phenomenon is cumulative prospect theory (CPT) [267], which is a generalization of expected utility and one of the leading decision theory models. It comprises of a probability weighting function in addition to the utility function that captures the probabilistic-sensitivity of the customers. Besides, such probabilistic-sensitivities are more commonly observed in small customers who are likely to adopt cloud-based services rather than maintaining their own computing systems.

In Section 10.2, we explain the CPT model in detail. We then formulate a system problem called SYS-CPT for a scheduler that allocates lotteries to the users based on their job requests and CPT features, and the capacity constraints from the cloud. We onserve that this problem is non-convex in general and it is hard to solve it optimally. We then use the discretization method proposed in [248]. The idea is to restrict the lotteries presented to the users to have their probabilities of with granularity $1/K$, where $K > 0$ is an integer. (see (10.16)). For example, when $K = 100$, we restrict all the probabilities to be of the form $Z\%$, where Z is an integer from 0 to 100. This gives rise to a discretized version of the system problem that we call SYS-CPT-K.

In Section 10.3, we analyse the problem SYS-CPT-K. We consider a relaxation to this problem that we call SYS-CPT-K-R. The problem SYS-CPT-K-R is LP and can be solved efficiently. We then establish several qualitative properties related to optimal solutions of SYS-CPT-K-R.

We then use these properties to show how to obtain an approximation to the optimal solution of the non-relaxed problem SYS-CPT-K and give theoretical bounds on this approximation. Although the number of variables in SYS-CPT-K-R are K times those in SYS-CPT-K, the qualitative results abtained on the optimal solutions allow us to restrict the number of variables significantly. This is important to implement and solve these problems in practice. It also allows us to conduct our experiments in section 10.4.

In Section 10.4, we decompose the problem SYS-CPT-K-R into several user problems, one for each user, and the cloud problem. This gives rise to a novel pricing mechanism that we discuss in Section 10.4. Using this pricing mechanism we simulate a market dynamics in Section 10.4.

10.2 Model

We first recall the model for serverless computing systems from [266]. According to this model, the service provider has access to a certain fleet of serverless workers, which it can use to execute the functions submitted by the customers (or users). We assume that each function execution requires one serverless worker and takes unit time (e.g., 100 ms). The number of serverless workers avaiblable to the service provider at any given time is an input to our model. A user can request for a job comprised of any number of executions of her function at any time (which could be triggered by external events.) We treat these requests as inputs to our system. We model the decision making task of the service provider, namely, the allocation of serverless workers to execute the submitted jobs, as a scheduler shown in figure 10.1.

The scheduler's decisions affect the quality of service of the users, in particular, the completion times of the user's submitted jobs. User i's job is said to be completed when all the J_i functions in her job are executed. We assume that the users have certain preferences over the delays in their job completion. Let $\mathcal{U}_i : \mathbb{R} \cup \{\infty\} \to \mathbb{R}$ denote the utility function of user i. Here, $\mathcal{U}_i(\tau)$ denotes the utility for user i corresponding to her job getting completed at time τ. We assume that $\mathcal{U}_i$ is a non-increasing function with $\mathcal{U}_i(\infty) = 0$. (See figure 9.1 from Chapter 9.) In contrast to the model in [266], we allow the delay in user i's job to be randomized and use lotteries to denote the underlying uncertainties. For example, consider the lottery

$$L_i := \{(p_i(1), y_i(1)), \ldots, (p_i(K), y_i(K))\}, \tag{10.1}$$

where $y_i(k) \in \mathbb{R}_+ \cup \{\infty\}, k \in [K]$, are the completion times and $p_i(k)$ the corresponding probabilities with which they occur. We assume the lottery to be exhaustive, i.e. $\sum_{j=1}^{K} p_i(k) = 1$. (Note that we are allowed to have $p_i(k) = 0$ for some values of $k \in [K]$ and $y_i(k) = y_i(k')$ for some $k, k' \in [K]$. A completion time of ∞ denotes that the job is not completed.) Thus, a lottery

$$\{(0.15, 0.1s); (0.05, 2s); (0.2, 10s); (0.6, 1000s)\}, \tag{10.2}$$

would mean that the user's job is completed at $\tau = 0.1s$ with a 15% chance, at $\tau = 2s$ with a 5% chance, at $\tau = 10s$ with a 20% chance, and at $\tau = 1000s$ with a 60% chance. Correspondingly, we note that the user's job is completed within $0.1s$ with a 15% chance, within $2s$ with a 20% chance, within $10s$ with a 40% chance, and within $1000s$ for sure, i.e. with a 100% chance.

We assume that the users have certain preferences over such delay-lotteries. According to expected utility theory, each user computes the expected utility from a given lottery. Thus the "happiness" of user i under expected utility theory (EUT) for lottery L_i would be

$$\sum_{k=1}^{K} p_i(k)\mathcal{U}_i(y_i(k)).$$

For example, the expected utility of user 2 in figure 9.1 (from Chapter 9) for the lottery L_i in (10.2) would be

$$0.15 \times 4 + 0.05 \times 3 + 0.2 \times 2.5 + 0.6 \times 1 = 1.85$$

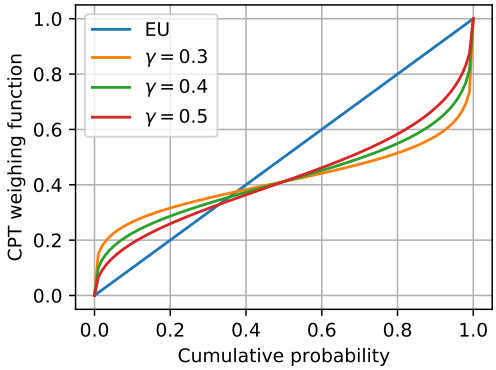


Figure 10.2: Examples of probability weighting functions. Here, we have used the form $w_i(p) = \frac{\delta p^\gamma}{\delta p^\gamma + (1-p)^\gamma}$, suggested by [269] with $\delta = 0.7$ and γ as indicated in the plot.

This is a special case of the cumulative prospect theory model to which we will generalize now. Let us describe the CPT model that we use to measure the "happiness" derived by each player from her lottery (for more details see [268]). Suppose an agent faces a lottery L_i as shown in (10.1). Each player i is associated with a utility function $\mathcal{U}_i : \mathbb{R}_+ \to \mathbb{R}_+$ as before, and a probability weighting function $w_i : [0,1] \to [0,1]$ that is continuous, strictly increasing and satisfies $w_i(0) = 0$ and $w_i(1) = 1$. (See, for example, figure 10.2.)

For the prospect L_i, let $\pi_i : [K] \to [K]$ be a permutation such that

$$\mathcal{U}_i(\bar{y}_i(1)) \geq \mathcal{U}_i(\bar{y}_i(2)) \geq \cdots \geq \mathcal{U}_i(\bar{y}_i(K)), \tag{10.3}$$

and

$$\bar{y}_i(k) := y_i(\pi_i(k)) \text{ for all } k \in [K]. \tag{10.4}$$

This is equivalent to the ordering

$$\bar{y}_i(1) \leq \bar{y}_i(2) \leq \cdots \leq \bar{y}_i(k),$$

since the utility function $\mathcal{U}_i(\cdot)$ is assumed to be non-increasing. The prospect L_i can equivalently be written as

$$L_i = \{(\bar{p}_i(1), \bar{y}_i(1)); \ldots; (\bar{p}_i(k), \bar{y}_i(k))\},$$

where $\bar{p}_i(k) := p_i(\pi_i(k))$ for all $k \in [K]$. The *CPT value* of prospect L_i for the agent is evaluated using the utility function $\mathcal{U}_i(\cdot)$ and the probability weighting function $w_i(\cdot)$ as follows:

$$V_i(L) := \sum_{k=1}^{K} d_i(p_i, \pi_i, k)\mathcal{U}_i(\bar{y}_i(k)), \tag{10.5}$$

where $d_i(p_i, \pi_i, k)$ are the *decision weights* given by $d_i(p_i, \pi_i, 1) := w_i(\bar{p}(1))$ and

$$d_i(p_i, \pi_i, k) := w_i(\bar{p}_i(1) + \cdots + \bar{p}_i(k)) - w_i(\bar{p}_i(1) + \cdots + \bar{p}_i(k-1)),$$

for $1 < k \leq K$. Although the expression on the right in equation (10.5) depends on the permutation π_i, one can check that the formula evaluates to the same value $V_i(L_i)$ as long as π_i satisfies (10.3) and (10.4). Say, in our above example, the user 2 satisfies a CPT weighing of $w_2(p) = \frac{\delta p^\gamma}{\delta p^\gamma + (1-p)^\gamma}$ with $\delta = 0.7$ and $\gamma = 0.4$ (see Fig. 10.2 for a plot of the weighting function). Thus, the CPT value of user 2 is given by

$$\begin{aligned}&(w_2(0.15) - w_2(0)) \times 4 + (w_2(0.2) - w_2(0.15)) \times 3\\ &+(w_2(0.4) - w_2(0.2)) \times 2.5 + (w_2(1) - w_2(0.4)) \times 1 = 2.026\end{aligned}$$

The CPT value of prospect L_i, can equivalently be written as

$$V_i(L_i) = \sum_{k=1}^{K} w_i\big(\sum_{s=1}^{k} \bar{p}_i(s)\big)\left[\mathcal{U}_i(\bar{y}_i(k)) - \mathcal{U}_i(\bar{y}_i(k+1))\right],$$

where $\bar{y}_i(K+1) := 0$. Thus the worst end time $\bar{y}_i(K)$ is weighted by $w_i(1) = 1$, and every increment in the value of the allocations, $\mathcal{U}_i(\bar{y}_i(k)) - \mathcal{U}_i(\bar{y}_i(k+1)), \forall k \in [K-1]$, is weighted by the probability weighting function of the probability of receiving an allocation at least equal to $\bar{y}_i(k)$. For user 2, the CPT value can, thus, be equivalently written as

$$\begin{aligned}&w_2(0.15) \times [4-3] + w_2(0.2) \times [3 - 2.5]\\ &+w_2(0.4) \times [2.5 - 1] + w_2(1) \times [1 - 0] = 2.026\end{aligned}$$

This completes the description of the CPT value computation of the users over the delay-lotteries. Note that a user's CPT value is completely characterized by her utility function and her probability weighting function.

We will now recall the scheduler model from [266], and modify it to allow for randomized resource allocations. A queue is maintained by the scheduler, where all the jobs requested by the users are added. A job remains in the queue until it is scheduled for execution. The scheduler is operated periodically and schedules the jobs that are currently in its queue. Let J_i denote the size of job i in the queue, i.e. the number of function executions needed for job i. (If there are two or more jobs corresponding to the same user in the queue, we will treat them as jobs from different users with identical characteristics.) Let $[N] := \{1, \ldots, N\}$ be the set of jobs that are currently pending in the queue. We will refer to job i as user i's job.

As in [266], we will consider the multi-tier model of resource allocation with T service tiers. Let τ_t denote the end time of tier t and $U_{i,t} := \mathcal{U}_i(\tau_t)$ be the utility of user i corresponding to her job getting completed in tier t. Let $P_{i,t}$ denote the probability that user i's jobs are completed by the end of tier t. For all i, t, let

$$p_{i,t} := P_{i,t} - P_{i,(t-1)} \tag{10.6}$$

denote the probability that user i's jobs are completed in tier t, i.e., they are completed by the end of tier t but not by the end of tier $(t-1)$.

Let M_t denote the number of available machines in tier t. Suppose we ask that the cloud capacity constraints are satisfied in expectation. Then this condition can be written as the constraint

$$\sum_{i=1}^{N} p_{i,t} J_i \leq M_t, \tag{10.7}$$

for all t. Suppose the cloud service provider does the following: For each job i and tier t it tosses a coin with a bias $p_{i,t}$. If it lands heads and the job i has not been executed before tier t then it is executed in tier t. As a result of such independent sampling, it can happen that the total required job executions in tier t exceeds the capacity M_t. Cloud service providers often have reserves which they can use in such cases. As a first order condition, we know from queuing theory that the system with such reserves will be stable if it satisfies the capacity contraints in expectation. Hence we consider the capacity constraints as in (10.7).

The problem of maximizing the total "happiness" of all the users with a probabilistic allocation of resources to users is given by

SYS-CPT

$$\max_{p_{i,t}, P_{i,t} \geq 0} \quad \sum_{i=1}^{N} \sum_{t=1}^{T} w_i\left(P_{i,t}\right) u_{i,t} \tag{10.8}$$

$$\text{subject to} \quad \sum_{t=1}^{T} p_{i,t} \leq 1, \forall i \in [N], \tag{10.9}$$

$$\sum_{i=1}^{N} p_{i,t} J_i \leq M_t, \forall t \in [T], \tag{10.10}$$

$$\sum_{s=1}^{t} p_{i,s} = P_{i,t}, \ \forall\, i \in [N], t \in [T], \tag{10.11}$$

where $u_{i,t} := U_{i,t} - U_{i,(t+1)}$.

If the users are restricted to have EUT preferences, then the system problem of maximizing the total expected utility for all the users with a probabilistic allocation of resources to users is given by

SYS-EU

$$\max_{p_{i,t} \geq 0} \quad \sum_{i=1}^{N} \sum_{t=1}^{T} p_{i,t} U_{i,t}$$

$$\text{subject to} \quad \sum_{t=1}^{T} p_{i,t} \leq 1, \forall i \in [N], \tag{10.12}$$

$$\sum_{i=1}^{N} p_{i,t} J_i \leq M_t, \forall t \in [T]. \tag{10.13}$$

We observe that SYS-EU is exactly same as SYS-LP in [266] with the transformation $x_{i,t} = p_{i,t}J_i$.

SYS-LP

$$\max_{x_{i,t}\geq 0} \quad \sum_{i=1}^{N}\sum_{t=1}^{T} x_{i,t}\frac{U_{i,t}}{J_i}$$

$$\text{subject to} \quad \sum_{t=1}^{T} x_{i,t} \leq J_i, \forall i \in [N], \tag{10.14}$$

$$\sum_{i=1}^{N} x_{i,t} \leq M_t, \forall t \in [T]. \tag{10.15}$$

Being a linear programming, SYS-LP can be solved efficiently, and hence we get the solution for SYS-EU. Further, it is shown in [266] that under certain natural conditions, the optimal solution $x_{i,t}$ to SYS-LP is such that there are at most T non-zero entries in the matrix $x_{i,t}$ that are not equal to J_i. Translating this into the optimal solution $p_{i,t}$, we get that except for at most T entries, all other jobs are either not executed at all or executed with probability 1 in some tier. That is, for most of the users we have a lottery L_i that allocates resources in a specific tier with probability 1. Thus, *for most of the users, we assign them deterministic lotteries under EUT.*

Remark 17. The problem SYS-CPT, in general, is non-convex and has several local minima. We verified this by solving SYS-CPT in the popular non-convex programming framework PyTorch [270]. We used gradient descent with back-propagation and observed that the solution reached different minima for different initializations of gradient descent.

We now use the discretization method introduced in [248] to solve SYS-CPT approximately. Besides, there are behavioral reasons to adopt such a discretization. We will explain them shortly. Let us first describe the discretization method. Fix an integer $K > 0$. We restrict our attention to lotteries where each user i is shown lotteries

$$\hat{L}_i = \{(q_{i,1}, \tau_1); \ldots; (q_{i,1}, \tau_T)\}$$

such that

$$q_{i,t} \in \Delta := \left\{\frac{k}{K} : 0 \leq k \leq K\right\}, \tag{10.16}$$

for all t. Let

$$Q_{i,t} := \sum_{s=1}^{t} q_{i,t}, \tag{10.17}$$

denote the corresponding cumulative probabilities for all t. Note that

$$Q_{i,t} \in \left\{\frac{k}{K} : 0 \leq k \leq K\right\} = \Delta, \tag{10.18}$$

for all t. As $K \to \infty$, we get better approximations to any probability distribution. However, there is a practical motivation to keep K limited. From a behavioral perspective, people often do not comprehend small differences in probability. For this reason, it is useful to restrict K (for example $K \leq 100$).

Although we restrict ourselves to presenting lotteries to the agents with probabilities belonging to the discretized mesh Δ, it does not stop us from using implementations with general probabilities as long as we are providing the agents at least what we are promising them. Let us make this argument concrete. Suppose we show each player i a lottery L_i with cumulative probabilities

$$Q_{i,\bullet} := (Q_{i,t})_{t \in [T]},$$

and corresponding probabilities

$$q_{i,\bullet} := (q_{i,t})_{t \in [T]},$$

satisfying (10.17). Suppose we use an implementation defined by probabilities $\mathbf{p} := (p_{i,t})_{i,t}$, where $p_{i,t}$ denotes the probability with which player i is allocated her resources J_i in tier t. Thus the true lottery for player i is given by

$$L_i := \{(p_{i,1}, \tau_1); \ldots; (p_{i,1}, \tau_T)\}.$$

Let

$$p_{i,\bullet} := (p_{i,t})_t,$$

and the corresponding cumulative probabilities

$$P_{i,\bullet} := (P_{i,t})_t,$$

satisfy (10.6)

We use the notion of first order stochastic dominance to determine when one lottery is better than the other. Lottery L_i first order stochastically dominates lottery $\hat{L}_i$ if

$$P_{i,t} \geq Q_{i,t}, \forall t \in [T].$$

For example, consider the lottery

$$\{(0.10, 0.1s); (0, 2s); (0.25, 10s); (0.65, 1000s)\}. \tag{10.19}$$

Compare this lottery with the lottery shown in (10.2). It is easy to check that the lottery shown in (10.2) first order stochastically dominates the lottery shown in (10.19). We notice that the lottery in (10.19) is formed by shifting a 5% chance from outcome $0.1s$ to outcome $1000s$ and a 5% chance from outcome $2s$ to $10s$. In general, if lottery L_i first order stochastically dominates lottery $\hat{L}_i$, then lottery L_i is formed by shifting probability mass from some outcomes to outcomes that are strictly preferred to them. This is a very strong condition of dominance and it is widely held that any satisfactory theory of preferences should satisfy it [271]. In particular, the CPT preferences and EUT preferences satisfy this condition [272].

We ask that $\mathbf{p}$ be implementable, i.e. it satisfies the capacity constraints (10.9) and (10.10). For each player i, lottery L_i corresonding to $p_{i,\bullet}$ first order stochastically dominates lottery $\hat{L}_i$ corresponding to $q_{i,\bullet}$. The probabilities $Q_{i,t}$ and $q_{i,t}$ belong to the mesh Δ. Let us call this discretized version of SYS-CPT as SYS-CPT-K, that is given by

SYS-CPT-K

$$\max_{p_{i,t}\geq 0, Q_{i,t}\in\Delta} \quad \sum_{i=1}^{N}\sum_{t=1}^{T} w_i\left(Q_{i,t}\right) u_{i,t} \tag{10.20}$$

$$\text{subject to} \quad \sum_{t=1}^{T} p_{it} \leq 1, \forall i \in [N] \tag{10.21}$$

$$\sum_{i=1}^{N} p_{i,t} J_i \leq M_t, \forall t \in [T], \tag{10.22}$$

$$\sum_{s=1}^{t} p_{i,s} \geq Q_{it}, \ \forall\, i \in [N], t \in [T]. \tag{10.23}$$

10.3 Analysis

We now give a richer problem formulation that would help us solve the above discretized version of SYS-CPT, namely, SYS-CPT-K. Suppose the cloud provider achieves a lottery implementation through a scheme as follows: The scheduler implements with equal probability one of the K allocation schemes

$$\mathbf{x}(k) := (x_{i,t}(k))_{i\in[N],t\in[T]},$$

for $k \in [K]$. Let $[K]$ denote the set of *alternatives*. For alternative k, the job end time for user i is given by

$$T_i(k) := \min\{t \in [T] : \sum_{s=1}^{t} x_{i,s}(k) \geq J_i\}, \tag{10.24}$$

We will follow the convention that the right hand side takes value $(T+1)$ if $\sum_{t=1}^{T} x_{i,t}(k) < J_i$. Thus, if the job is not completed by the end of tier T, then we say that it is completed in the dummy tier $(T+1)$. We will assume that $\tau_{(T+1)} = \infty$, and hence, $U_{i,(T+1)} = 0$, for all players i. Thus the probability that user i's job is completed by the end of tier t is given by

$$Q_{i,t} = \frac{|\{k : T_i(k) \leq t\}|}{K}, \tag{10.25}$$

and the lottery faced by user i is given by

$$L_i = \{(q_{i,1}, \tau_1); \ldots; (q_{i,T}, \tau_T)\},$$

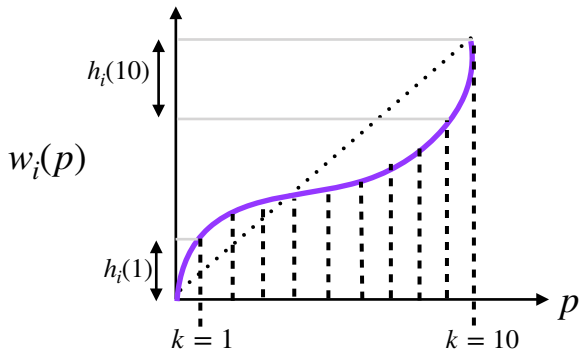


Figure 10.3: An illustration of how $h_i(\cdot)$ is calculated using the CPT weighing function $w_i(\cdot)$.

where

$$q_{i,t} = Q_{i,t} - Q_{i,(t-1)}. \tag{10.26}$$

Note that $q_{i,t}, Q_{i,t} \in \Delta$ for all i, t. The CPT value of this lottery is

$$V_i(L_i) = \sum_{t=1}^{T} w_i(Q_{i,t}) u_{i,t}. \tag{10.27}$$

This lottery can equivalent be written as

$$L_i = \{(1/K, \tau_{T_i(1)}); \ldots; (1/K, \tau_{T_i(k)})\}.$$

Note that in this formulation the end times can appear repeatedly, but the total probability of any particular end time is same in both the formulations. It is easy to verify that the CPT value formula is invariant under such equivalent representations. Also note that the end times in the second formulation are not ordered. Let $\pi_i : [K] \to [K]$ be a permutation such that

$$\bar{T}_i(1) \leq \bar{T}_i(2) \leq \cdots \leq \bar{T}_i(K),$$

where $\bar{T}_i(k) = T_i(\pi_i(k))$, for all $k \in [K]$. Then the CPT value of user i for lottery L_i is given by

$$V_i(L_i) = \sum_{k=1}^{K} h_i(k) U_{i,\bar{T}_i(k)}. \tag{10.28}$$

where

$$h_i(k) := w_i(k/K) - w_i((k-1)/K),$$

for all $k \in [K]$. Note that the two formulas for the CPT value in (10.27) and (10.28) are equivalent. Also note that if $w_i(\cdot)$ is an identity function (as in the EUT setting), then $h_i(k) = 1/K$, and the corresponding CPT value is simply the average of the utilities from all the alternatives. An illustration of how $h_i(k)$'s are calculated is provided in Fig. 10.3.

We now use the relaxation similar to that used in [266]. We approximate the utility for user i corresponding to alternative k as follows:

$$U_{i,T_i(k)} \approx \sum_{t=1}^{T} \frac{x_{it}(k)U_{i,t}}{J_i}. \tag{10.29}$$

Note that if $(x_{it}(k), t \in [T])$ is 1-sparse, with that non-zero entry equal to J_i, or an all zero vector, then the above approximation is exact. With this relaxation our objective can be written as

$$\sum_{i=1}^{N}\sum_{k=1}^{K} h_i(k)\bar{F}_i(x_{i,\bullet}(\pi_i(k))),$$

where $\bar{F}(x_{i,\bullet}(k)) := \sum_{t=1}^{T} x_{i,t}(k)F_{i,t} = \sum_{t=1}^{T} x_{i,t}(k)U_{i,t}/J_i$ and the permutation $\pi_i(k)$ is such that

$$\bar{F}_i(x_{i,\bullet}(\pi_i(1))) \geq \bar{F}_i(x_{i,\bullet}(\pi_i(2))) \geq \cdots \geq \bar{F}_i(x_{i,\bullet}(\pi_i(K))), \tag{10.30}$$

for all i.

Corresponding to allocations $\mathbf{x}(k), \forall k \in [K]$ and permutations $\pi_i, \forall i \in [N]$, let

$$z_{i,t}(k) := x_{i,t}(\pi_i(k))$$

be a $|T|$-dimensional column vector, for $i \in [N], k \in [K]$. With this notation, the objective function can be written as

$$\sum_{i=1}^{N}\sum_{k=1}^{K} h_i(k)\bar{F}_i(z_{i,\bullet}(k)), \tag{10.31}$$

and the inequalities (10.30) take the form

$$\bar{F}_i(z_{i,\bullet}(1)) \geq \bar{F}_i(z_{i,\bullet}(2)) \geq \cdots \geq \bar{F}_i(z_{i,\bullet}(k)), \tag{10.32}$$

for all i.

Strictly speaking, such a scheme in our richer problem formulation is feasible if and only if each of the alternatives is feasible, i.e.

$$\sum_{i=1}^{N} x_{it}(k) \leq M_t, \tag{10.33}$$

for all t, k. However, we will use this richer formulation only as an intermediate step to solve SYS-CPT-K, where we are interested in the implementations that are feasible in expectation. Hence we will, instead, consider the constraint

$$\frac{1}{K}\sum_{k=1}^{K}\sum_{i=1}^{N} x_{i,t}(k) \leq M_t, \forall t \in [T]. \tag{10.34}$$

This motivates the following relaxed system problem:

SYS-CPT-K-R

$$\max_{\mathbf{z}(k)\geq 0} \quad \sum_{i=1}^{N}\sum_{k=1}^{K} h_i(k)\bar{F}_i(z_{i,\bullet}(k)) \tag{10.35}$$

$$\text{subject to} \quad \sum_{t=1}^{T} z_{i,t}(k) \leq J_i,\ \forall\, i \in [N], k \in [K], \tag{10.36}$$

$$\frac{1}{K}\sum_{k=1}^{K}\sum_{i=1}^{N} z_{i,t}(k) \leq M_t,\ \forall\, t \in [T], \tag{10.37}$$

$$\bar{F}_i(z_{i,\bullet}(k)) \geq \bar{F}_i(z_{i,\bullet}(k+1)),\ \forall\, i \in [N],\ \forall\, 1 \leq k \leq K-1. \tag{10.38}$$

Notice that the problem SYS-CPT-K-R is an LP problem. As compared to the previous problems SYS-CPT and SYS-CPT-K, the number of variables in SYS-CPT-K-R has become K-fold. Although this might seem like a problem from a computational point of view, later we will derive some qualitative features about the optimal solution which will let us significantly reduce the dimensions of our variables. But more on this later. Let us first observe how the solution to SYS-CPT-K-R compares to the solution to SYS-CPT-K.

Suppose $p^*_{i,t}, Q^*_{i,t}, \forall i, t$, is an optimal solution to the SYS-CPT-K problem. Then, let us construct a corresponding lottery allocation $(\mathbf{z}^*(k))_{k\in[K]}$ as follows: let

$$z^*_{i,t}(k) = \begin{cases} J_i, & \text{if } Q^*_{i,(t-1)} < k/K \leq Q^*_{i,t}, \\ 0, & \text{otherwise,} \end{cases} \tag{10.39}$$

for all i, t, k. (Recall that $Q^*_{i,t} \in \Delta, \forall i, t$.) One can verify that $(\mathbf{z}^*(k))_k$ is a feasible solution to SYS-CPT-K-R, and the objectives (10.35) and (10.8) match exactly. This is because the allocations $z^*_{i,\bullet}(k)$ are either 1-sparse with the non-zero entry equal to J_i, or equal to the zero vector, and hence the approximation in (10.29) is exact. Further, the optimal value of SYS-CPT-K matches

$$V^* := \sum_{i=1}^{N}\sum_{k=1}^{K} h_i(k)\bar{F}_i(z^*_{i,\bullet}(k)). \tag{10.40}$$

On the other hand, let $(\mathbf{z}^R(k))_k$ be an optimal solution of SYS-CPT-K-R. Let V^R denote the optimal value of the objective function of SYS-CPT-K-R, i.e.

$$V^R := \sum_{i=1}^{N}\sum_{k=1}^{K} h_i(k)\bar{F}_i(z^R_{i,\bullet}(k)). \tag{10.41}$$

We now define the end times $\tilde{T}_i(k)$, for each player i, for her kth best alternative by

$$\tilde{T}_i(k) := \min\{t \in [T] : \sum_{\tau=1}^{t} z^R_{i,\tau}(k) \geq J_i\}, \tag{10.42}$$

similarly to (10.24). Now consider $\tilde{Q}_{it}$ given by

$$\tilde{Q}_{i,t} := \frac{|\{k : \tilde{T}_i(k) \leq t\}|}{K}, \tag{10.43}$$

similar to (10.25). Finally, let

$$\tilde{z}_{i,t}(k) := \begin{cases} J_i, & \text{if } \tilde{Q}_{i,(t-1)} < k/K \leq \tilde{Q}_{i,t}, \\ 0, & \text{otherwise,} \end{cases} \tag{10.44}$$

similarly to (10.39). Let the value of the objective function in SYS-CPT-K-R corresponding to $\tilde{z}$ be

$$\tilde{V} := \sum_{i=1}^{N} \sum_{k=1}^{K} h_i(k) \bar{F}_i(\tilde{z}_{i,\bullet}(k)). \tag{10.45}$$

Now note that in the allocation $(\tilde{\mathbf{z}}(k))_k$, the players get resources only in their completion tier. Thus, they do not get any partial utility in our approximation (10.29). Hence, we have that $\tilde{V} \leq V^R$. Further, $\tilde{V}$ is equal to the value of the objective function in SYS-CPT-K corresponding to $\tilde{Q}_{i,t}$, i.e.

$$\tilde{V} = \sum_{i=1}^{N} \sum_{t=1}^{T} w_i \left(\tilde{Q}_{i,t} \right) u_{i,t}. \tag{10.46}$$

This is because the allocations $\tilde{z}_{i,\bullet}(k)$ are either 1-sparse with the non-zero entry equal to J_i, or equal to the zero vector. Now, if we let

$$p^R_{i,t} := \frac{1}{K} \sum_{k=1}^{K} z^R_{i,t}(k), \tag{10.47}$$

then $p^R_{i,t}, \tilde{Q}_{it}$ is a feasible solution to SYS-CPT-K with value $\tilde{V}$. Thus $\tilde{V} \leq V^*$, since V^* is an optimal solution to SYS-CPT-K.

Together, we get that

$$\tilde{V} \leq V^* \leq V^R. \tag{10.48}$$

In Theorem 15, we bound the gap between V^R and $\tilde{V}$ and thus get a bound on the gap between the optimal solution to SYS-CPT-K and the solution arising from our relaxation.

We will now prove certain qualitative features of the optimal solution to SYS-CPT-K-R that we had promised earlier. These will shed light on the optimal lottery allocations and help us bound the gap between the optimal solutions to SYS-CPT-K and SYS-CPT-K-R. Besides, it will significantly reduce the dimension of our problem SYS-CPT-K-R making it practical for actual deployment. This is what allows us to run our simulations.

To describe these qualitative features let us define a few parameters of interest related to the probability weighting functions $w_i, \forall i$. First we impose some additional conditions on the probability weighting functions that are typically based on empirical evidence and psychological

arguments [273]. We assume that the probability weighting function $w_i(p_i)$ is concave for small values of the probability p_i, say for $p_i \in [0, \tilde{p}_i]$, and convex for the rest. Typically the point of inflection is around $1/3$. Let $w_i^* : [0,1] \to [0,1]$ be the minimum concave function that dominates w_i, i.e. $w_i^*(p_i) \geq w_i(p_i)$ for all $p_i \in [0,1]$. Let $p_i^* \in [0,1]$ be the smallest probability such that $w_i^*(p_i)$ is linear over the interval $[p_i^*, 1]$. The following properties are evident from the figure (see [248] for their proofs). We have $p_i^* \leq \tilde{p}_i$, and

$$w_i^*(p_i) = w_i(p_i) \text{ for } p_i \in [0, p_i^*].$$

Further, if $p_i^* < 1$, then for any $p_i' \in [p_i^*, 1)$, we have

$$w_i(p_i) \leq w_i(p_i') + (p_i - p_i')\frac{1 - w_i(p_i')}{1 - p_i'},$$

for all $p_i \in [p_i', 1]$.

Proposition 14. *If $(\mathbf{z}(k))_k$ is an optimal solution to SYS-CPT-K-R, then*

$$\bar{F}_i(z_{i,\bullet}(k_i^*)) = \bar{F}_i(z_{i,\bullet}(k_i^* + 1)) = \cdots = \bar{F}_i(z_{i,\bullet}(K)), \tag{10.49}$$

where $k_i^* := \min\{k \in [K] : (k-1)/K \geq p_i^*\}$, provided $p_i^* \leq (K-1)/K$.

We prove this proposition in Appendix 10.6.1.

Additionally, if there is an optimal solution $(\mathbf{z}(k))_k$ such that $\bar{F}_i(z_{i,\bullet}(k)) = \bar{F}_i(z_{i,\bullet}(k+1))$, then we can construct a solution that is also optimal and satisfies $z_{i,\bullet}(k) = z_{i,\bullet}(k+1)$. To see this, consider the allocation $(\tilde{\mathbf{z}}(k))_k$ such that

$$\tilde{z}_{i,\bullet}(k) = \tilde{z}_{i,\bullet}(k+1) = \frac{z_{i,\bullet}(k) + z_{i,\bullet}(k+1)}{2},$$

and let $\tilde{\mathbf{z}}(k)$ be the same as $\mathbf{z}(k)$ everywhere else. It is easy to see that $\tilde{\mathbf{z}}$ is also an optimal solution to SYS-CPT-K-R.

Corollary 3. *There exists an optimal solution $(\mathbf{z}(k))_k$ to SYS-CPT-K-R such that* (10.49) *holds for all users i, and whenever $\bar{F}_i(z_{i,\bullet}(k)) = \bar{F}_i(z_{i,\bullet}(k+1))$, we have $z_{i,\bullet}(k) = z_{i,\bullet}(k+1)$.*

In the following, we will first establish some additional properties related to the structure of an optimal solution $(\mathbf{z}(k))_k$ to SYS-CPT-K-R, and use this to bound the gap between the optimal solutions of SYS-CPT-K and SYS-CPT-K-R.

The Lagrangian for the problem SYS-CPT-K-R is given by

$$\begin{aligned}
&L((\mathbf{z}(k))_k, (\lambda_i(k))_{i,k}, (\mu_t)_t, (\alpha_i(k))_k) \\
&\qquad = \sum_{i=1}^{N}\sum_{k=1}^{K} h_i(k)\bar{F}_i(z_{i,\bullet}(k)) - \sum_{i=1}^{N}\sum_{k=1}^{K}\lambda_i(k)\left(J_i - \sum_{t=1}^{T} z_{i,t}(k)\right) \\
&\qquad - \sum_{t=1}^{T}\mu_t\left(M_t - \frac{1}{K}\sum_{i=1}^{N}\sum_{k=1}^{K} z_{i,t}(k)\right) \\
&\qquad - \sum_{i=1}^{N}\sum_{k=1}^{K}\alpha_i(k)\left(\sum_{t=1}^{T} F_{i,t}z_{i,t}(k) - \sum_{t=1}^{T} F_{i,t}z_{i,t}(k+1)\right),
\end{aligned}$$

where $\lambda_i(k) \geq 0, \mu_t \geq 0, \alpha_i(k) \geq 0$ are the dual variables corresponding to the constraints (10.36), (10.37), and (10.38). The complementary slackness conditions for the SYS-CPT-K-R problem are:

$$h_i(k)F_{i,t} - \lambda_i(k) - \frac{\mu_t}{k} + F_{i,t}(\alpha_i(k) - \alpha_i(k-1)) \begin{cases} = 0, & \text{if } z_{i,t}(k) > 0, \\ \geq 0, & \text{if } z_{i,t}(k) = 0, \end{cases} \qquad \forall i, t, \tag{10.50}$$

$$\sum_{t=1}^{T} z_{i,t}(k) \begin{cases} = J_i, & \text{if } \lambda_i(k) > 0, \\ \leq J_i, & \text{if } \lambda_i(k) = 0, \end{cases} \qquad \forall i, k, \tag{10.51}$$

$$\frac{1}{K}\sum_{i=1}^{N}\sum_{k=1}^{K} z_{i,t}(k) \begin{cases} = M_t, & \text{if } \mu_t > 0, \\ \leq M_t, & \text{if } \mu_t = 0, \end{cases} \qquad \forall t, \tag{10.52}$$

$$\sum_{t=1}^{T} F_{i,t}z_{i,t}(k) - \sum_{t=1}^{T} F_{i,t}z_{i,t}(k+1) \begin{cases} = 0, & \text{if } \alpha_i(k) > 0, \\ \geq 0, & \text{if } \alpha_i(k) = 0, \end{cases} \qquad \forall i, k. \tag{10.53}$$

Now observe that $\alpha_i(0) = \alpha_i(k) = 0$ for all i. Consider the set $\mathcal{S}_i$ comprising of elements $(k_1, k_2; i)$ such that $0 < k_1 \leq k_2 < K$, $\alpha_i(k_1 - 1) = \alpha_i(k_2) = 0$, and $\alpha_i(k) > 0$ for all $k_1 \leq k \leq k_2 - 1$. Note that each $k \in [K]$ belongs to the interval defined by a unique element $(k_1, k_2; i) \in \mathcal{S}_i$. In words, k_1 and k_2 determine the maximal contiguous strip of indices such that $\alpha_i(k) > 0$. From (10.53), $\alpha_i(k) > 0$ implies $\sum_{t=1}^{T} F_{i,t}z_{i,t}(k) = \sum_{t=1}^{T} F_{i,t}z_{i,t}(k+1)$.

As observed above, without loss of generality, let $(\mathbf{z}(k)_k)$ be an optimal solution such that whenever $\bar{F}_i(z_{i,\bullet}(k)) = \bar{F}_i(z_{i,\bullet}(k+1))$, we have $z_{i,\bullet}(k) = z_{i,\bullet}(k+1)$. Thus, corresponding to any element $(k_1, k_2; i) \in \mathcal{S}_i$, we have a common vector $z^*_{i,\bullet} := (z^*_{it})_{t\in[T]}$ such that $z^*_{i,\bullet} = z_{i,\bullet}(k)$, for all $k_1 \leq k \leq k_2$. From (10.50), we have

$$h_i(k)F_{i,t} - \lambda_i(k) - \frac{\mu_t}{k} = F_{i,t}(\alpha_i(k-1) - \alpha_i(k)), \tag{10.54}$$

for all $z^*_{i,t} > 0$. Telescoping over $k_1 \leq k \leq k_2 + 1$, we get

$$F_{i,t}\left(\sum_{k=k_1}^{k_2} h_i(k)\right) = \sum_{k=k_1}^{k_2}\lambda_i(k) + \sum_{k=k_1}^{k_2}\frac{\mu_t}{k}. \tag{10.55}$$

Suppose there are more than one tiers t, say $t \neq t'$, for which $z^*_{i,t} > 0$, then we get

$$(F_{i,t} - F_{i,t'}) \left(\sum_{k=k_1}^{k_2} h_i(k) \right) = (k_2 - k_1 + 1) \frac{\mu_t - \mu_{t'}}{K}. \tag{10.56}$$

Thus, if we have an intance $(k_1, k_2, t, t'; i)$ such that $(k_1, k_2; i) \in \mathcal{S}_i$ and $z^*_{i,t} > 0$ and $z^*_{i,t'} > 0$ for the corresponding common allocation vector $z^*_{i,\bullet}$, then we have an equality relation as shown in (10.56). We now note that the parameters μ_t are not indexed by i. If we have more than T such distinct instances then we get a non-trivial relationship between the variables $F_{i,t}$. Now using an argument similar to the one given in [266], we have

Lemma 16. *There exists an optimal solution $(\mathbf{z}(k))_k$ such that except for at most T elements in $\cup_i \mathcal{S}_i$, the corresponding common vector $z^*_{i,\bullet}$ has either one element that is equal to J_i, or is the zero vector.*

Let $\mathcal{T} \subset \cup_i \mathcal{S}_i$ denote the set of all elements $(k_1, k_2; i)$ such that the corresponding common vector $z^*_{i,\bullet}$ either has more than one non-zero elements or has a single non-zero element not equal to J_i. According to lemma 16, there exists an optimal solution $(\mathbf{z}(k))_k$ such that $|\mathcal{T}| \leq T$. The proof is similar to the one in [266] and we omit it. Using this property regarding the existence of a sparse solution, we get a bound on the gap that we prove in appendix 10.6.2.

Theorem 15. *Let V^* and V^R be the optimal objective values of the problem in SYS-CPT-K and SYS-CPT-K-R, respectively. We have,*

$$V^* \geq \left(1 - \frac{T(\max_i J_i)}{\min_t M_t} \right) V^R. \tag{10.57}$$

10.4 Pricing Mechanisms for Lottery-based Allocation

Previously, we assumed that the users' utilities are known at the cloud provider. However, this is not true in general. Here, we design a feedback-based scheme to find the optimal lottery allocation. We will decompose the system problem SYS-CPT-K-R into USER(i) problems–one for each user–and a CLOUD problem.

First, let us see how the observation in Corollary 3 helps us reduce the dimensions of the problem SYS-CPT-K-R. For each player i, we have $p^*_i \in [0, 1]$ as defined above Proposition 14. As noted earlier, $p^*_i \leq 1/3$ for typical probability weighting functions. Let $K \geq 3$. Thus, $p^*_i \leq (K-1)/K$. Let $k^*_i = \min\{k \in [K] : (k-1)/K \geq p^*_i\}$, for each player i. Let $k^* = \max_i\{k^*_i\}$. From Proposition 3, we know that there exists an optimal solution to SYS-CPT-K-R such that the lottery allocations $z_{i,\bullet}(k)$ are identical for all $k \geq k^*$. Let us club all these alternatives into a single alternative K^*. We consider a setting with $K^* = k^*$ alternatives, where each of the alternatives $k \in [K^* - 1]$ has its corresponding probability $1/K$ and the last alternative K^* has its corresponding

probability $(K - K^* + 1)/K$. Let

$$\delta(k) := \begin{cases} 1/K, & \text{for } 1 \leq k < K^*, \\ (K - K^* + 1)/K, & \text{for } k = K^*. \end{cases}$$

Let

$$h_i^*(k) := \begin{cases} w_i(k/K) - w_i((k-1)/K), & \text{for } 1 \leq k < K^*, \\ 1 - w_i(k^* - 1/K), & \text{for } k = K^*. \end{cases}$$

Now, without loss of generality, we can reduce the problem SYS-CPT-K-R to the following problem:

SYS-CPT-K*-R

$$\max_{\mathbf{z}(k)\geq 0} \quad \sum_{i=1}^{N}\sum_{k=1}^{K^*} h_i^*(k)\bar{F}_i(z_{i,\bullet}(k)) \tag{10.58}$$

$$\text{subject to} \quad \sum_{t=1}^{T} z_{i,t}(k) \leq J_i, \ \forall\, i \in [N], k \in [K^*], \tag{10.59}$$

$$\sum_{k=1}^{K^*}\sum_{i=1}^{N} \delta(k) z_{i,t}(k) \leq M_t, \ \forall\, t \in [T], \tag{10.60}$$

$$\bar{F}_i(z_{i,\bullet}(k)) \geq \bar{F}_i(z_{i,\bullet}(k+1)), \ \forall\, i \in [N], \ \forall\, 1 \leq k \leq K^* - 1. \tag{10.61}$$

To decompose the problem, we define a new variable matrices $m_{i,\bullet}(k) \in \mathbb{R}^T$, for all $i \in [N], k \in [K^*]$, which is representative of user i's budgets corresponding to different alternatives k and different tiers t. Let $(\mu_t)_{t\in[T]}$ denote a price vector.

We define the USER(i) problem as follows:

USER (i)

$$\max_{\substack{m_{i,\bullet}(k),\forall k\in[K^*],\\ m_{i,t}(k)\geq 0,\forall t,k}} \quad \sum_{t=1}^{T}\sum_{k=1}^{K^*} \frac{m_{i,t}(k)}{\mu_t}\left(h_i^*(k)F_{i,t} - \delta(k)\mu_t\right)$$

$$\text{subject to} \quad \sum_{t=1}^{T} \frac{m_{i,t}(k)}{\mu_t} \leq J_i, \ \forall\, k \in [K^*], \tag{10.62}$$

$$\sum_{t=1}^{T} F_{i,t}\frac{m_{i,t}(k)}{\mu_t} \geq \sum_{t=1}^{T} F_{i,t}\frac{m_{i,,t}(k+1)}{\mu_t}, \ \forall\, 1 \leq k \leq K^* - 1. \tag{10.63}$$

Given the budget vectors $m_i(k) \geq 0$ for all players i and alternatives k, consider the following problem for the cloud service provider:

CLOUD

$$\begin{aligned} \max_{z_{i,t}(k)\geq 0} \quad & \sum_{k=1}^{K^*}\sum_{i=1}^{N}\sum_{t=1}^{T}\delta(k)m_{i,t}(k)\log(z_{i,t}(k)) \\ \text{subject to} \quad & \sum_{k=1}^{K^*}\sum_{i=1}^{N}\delta(k)z_{i,t}(k) \leq M_t,\ \forall\, t\in[T]. \end{aligned} \tag{10.64}$$

Theorem 16. *There exist an equilibrium price vector $(\mu_t)_t$, allocation matrices $\mathbf{z}(k)$ for all $k \in [K^*]$, and budget vectors $m_{i,\bullet}(k)$ for all $i \in [N], k \in [K^*]$, such that*

1. *$(m_{i,\bullet}(k))_{k\in[K^*]}$ solves USER(i), $\forall\, i \in [N]$,*
2. *$(\mathbf{z}(k))_{k\in[K^*]}$ solves the CLOUD problem,*
3. *$m_{i,t}(k) = z_{i,t}(k)\mu_t$, for all $i \in [N], t \in [T], k \in [K]$,*
4. *for any t, if $\sum_i \sum_k \delta(k)z_{i,t}(k) < M_t$, then $q_t = 0$.*

Further, if the allocations $(\mathbf{z}(k))_k$ are at equilibrium, i.e. have a corresponding price vector $(\mu_t)_t$ and budget vectors $m_{i,\bullet}(k)$ for all $i \in [N], k \in [K^*]$, that together satisfy (i), (ii), (iii), and (iv), then $(\mathbf{z}(k))_k$ solves the problem SYS-CPT-K-R.*

Proof. Similarly to the Lagrangian for SYS-CPT-K-R, we can form the Lagrangian for SYS-CPT-K*-R with the dual variables $\lambda_i(k) \geq 0, i \in [N], k \in [K^*]$, $\mu_t \geq 0, t \in [T]$, and $\alpha_i(k) \geq 0, i \in [N], k \in [K^*]$. The corresponding complementary slackness conditions take the form

$$h_i^*(k)F_{i,t} - \lambda_i(k) - \frac{\mu_t}{k} + F_{i,t}(\alpha_i(k) - \alpha_i(k-1)) \begin{cases} = 0, & \text{if } z_{i,t}(k) > 0, \\ \geq 0, & \text{if } z_{i,t}(k) = 0, \end{cases} \qquad \forall i,t,k \in [K^*] \tag{10.65}$$

$$\sum_{t=1}^{T} z_{i,t}(k) \begin{cases} = J_i, & \text{if } \lambda_i(k) > 0, \\ \leq J_i, & \text{if } \lambda_i(k) = 0, \end{cases} \qquad \forall i,k, \tag{10.66}$$

$$\sum_{i=1}^{N}\sum_{k=1}^{K} \delta(k)z_{i,t}(k) \begin{cases} = M_t, & \text{if } \mu_t > 0, \\ \leq M_t, & \text{if } \mu_t = 0, \end{cases} \qquad \forall t, \tag{10.67}$$

$$\sum_{t=1}^{T} F_{i,t}z_{i,t}(k) - \sum_{t=1}^{T} F_{i,t}z_{i,t}(k+1) \begin{cases} = 0, & \text{if } \alpha_i(k) > 0, \\ \geq 0, & \text{if } \alpha_i(k) = 0, \end{cases} \qquad \forall i,k. \tag{10.68}$$

The Lagrangian for the USER(i) problem with $\gamma_i(k)$ and $\beta_i(k)$, $i \in [N], k \in [K]$ as dual variables corresponding to the two constraints (10.62) and (10.63) is given by

$$\begin{aligned}
&L((m_{i,\bullet}(k))_k, (\gamma_i(k))_k, (\beta_i(k))_k) \\
&= \sum_{k=1}^{K^*} \frac{m_{i,t}(k)}{\mu_t}(h_i^*(k)F_{i,t} - \delta(k)\mu_t) \\
&+ \sum_{k=1}^{K^*} \gamma_i(k) \left(J_i - \frac{m_{i,t}(k)}{\mu_t} \right) + \sum_{k=1}^{K^*-1} \beta_i(k) \left(\sum_{t=1}^{T} \frac{m_{i,t}(k)}{\mu_t} F_{i,t} - \sum_{t=1}^{T} \frac{m_{i,t}(k+1)}{\mu_t} F_{i,t} \right).
\end{aligned}$$

The corresponding KKT conditions are given by

$$h_i^*(k)F_{i,t} - \gamma_i(k) - \delta(k)\mu_t + F_{i,t}(\beta_i(k) - \beta_i(k-1)) \begin{cases} = 0, & \text{if } m_{i,t}(k) > 0, \\ \geq 0, & \text{if } m_{i,t}(k) = 0, \end{cases} \quad \forall i, t, k \in [K^*], \tag{10.69}$$

$$\sum_{t=1}^{T} \frac{m_{i,t}(k)}{\mu_t} \begin{cases} = J_i, & \text{if } \gamma_i(k) > 0, \\ \leq J_i, & \text{if } \gamma_i(k) = 0, \end{cases} \quad \forall i, k, \tag{10.70}$$

$$\sum_{t=1}^{T} \frac{F_{i,t}}{\mu_t}(m_{i,t}(k) - m_{i,t}(k+1)) \begin{cases} = 0, & \text{if } \beta_i(k) > 0, \\ \geq 0, & \text{if } \beta_i(k) = 0, \end{cases} \quad \forall i, k. \tag{10.71}$$

Similarly, for the CLOUD problem, the Lagrangian with dual variables $q_t, t \in [T]$ can be written as

$$L((\mathbf{z}(k)_k, (q_t)_t) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K^*} \delta(k) m_{i,t}(k) \log(z_{i,t}(k)) + \sum_{t=1}^{T} q_t \left(M_t - \sum_{k=1}^{K^*} \sum_{i=1}^{N} \delta(k) z_{i,t}(k) \right).$$

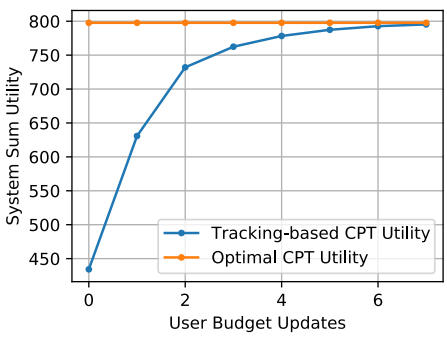
The corresponding KKT conditions are given by

$$m_{i,t}(k) = q_t z_{i,t}(k) \quad \forall \quad i, t, k \in [K^*], \tag{10.72}$$

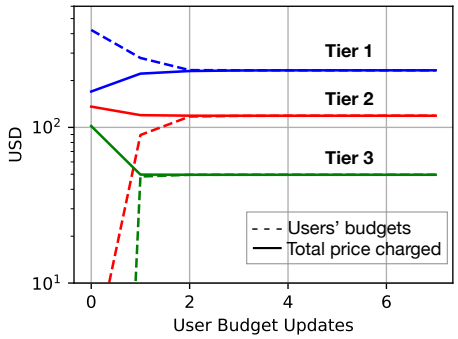
$$\sum_{i=1}^{N} \sum_{k=1}^{K} \delta(k) z_{i,t}(k) \begin{cases} = M_t, & \text{if } q_t > 0, \\ \leq M_t, & \text{if } q_t = 0, \end{cases} \quad \forall t, \tag{10.73}$$

Now, observe that given a solution $(\mathbf{z}, \lambda, \mu, \alpha)$ to SYS-CPT-K-R and satisfies the corresponding KKT conditions, it also satisfies the KKT conditions for USER(i), for all $i \in [N]$, and CLOUD problems with $m_{it}^k = z_{it}^k/\lambda, \mathbf{q} = \boldsymbol{\mu}, \mathbf{p} = \boldsymbol{\lambda}$ and $\boldsymbol{\beta} = \alpha$. □

The detailed steps of the pricing mechanism process is provided in Algorithm 21. Further, in Fig. 10.4, we plot the results of decentralized pricing scheme from Algorithm 21 for $N = 100$



(a) The tracking algorithm eventually achieves optimal utility.



(b) The prices set by the cloud provider and the budgets showed by the users converge to the optimal with the tracking algorithm.

Figure 10.4: Performance of the tracking-based algorithm.

users and $T = 3$ tiers with i.i.d. user utilities. CPT weighing function is given shown in Fig. 10.2 where γ is chosen i.i.d. from $[0.3, 0.5]$ for each user. Number of gradient steps, G, was chosen to be 100 and the step-size κ was kept at 3×10^{-5}. From Fig. 10.4a, we note that after only 7 iterations (budget/pricing) updates, the system utility of the tracking-based scheme converges to the optimal system utility. Further, as seen from Fig. 10.4b, with small changes in the pricing scheme, the cloud service provider is able to nudge the users to change their budget to match the optimal solution.

10.5 Experiments

In this section, we present some experiments to show the advantages of considering a CPT based utility objective over a naive Expected Utility (EU) objective considered earlier. We consider the general CPT weighing function from Figure 10.2 that is a function of two parameters, that is, for the i-th user, the CPT weighing is given by

$$w_i\left(p_i\right) = \frac{\delta p_i^{\gamma_i}}{\delta p^{\gamma_i} + (1-p)^{\gamma_i}}.$$

For all the experiments in this section, we keep $\delta = 0.7$ and γ_i is chosen uniformly randomly in an i.i.d. fashion from $[0.3, 0.5]$ for each user i. Further, we fix the number of users $N = 100$, the number of service tiers $T = 3$, and the number of machines at the cloud provider $M_t = 1000$, for all t. Each user obtains a utility between 50 and 100, uniformly chosen, upon successful completion of all her jobs. In Fig. 10.5, we plot the variation in utilities when the demand of the users is varied. The number of jobs at the i-th user, J_i, is chosen to be a uniform random variable between 10 and 99 for all $i \in [1, 100]$. To vary the demand in the system, the number of users in the system is increased from 20 to 100, and the overall utility of the system with the CPT and EUT objectives is plotted.

We see the when the number of users is increased (i.e., the demand is increased while keeping the supply constant), the system utility with the CPT objective increases. In particular, when the

Algorithm 21: Finding the optimal resource allocation and pricing without utility information at the cloud end

Input : Total number of machines M, step-size κ, error tolerance ϵ, gradient steps G, job sizes J_i, CPT weighing $w_i(\cdot)$, and utilities $U_i(\cdot)$ (known only to user i for $i \in [N]$)

Initialization: Cloud service provider shows some initial prices $\mathbf{q} = \mathbf{q}_0 \in \mathbb{R}^T$ (say, the all ones vector)

while $\|\mathbf{q} - \mathbf{q}_{prev}\| \geq \epsilon \|\mathbf{q}_{prev}\|$ **do**

  $\mathbf{q}_{prev} = \mathbf{q}$

  Users, for all, $i \in [N]$, solve for budget vector $\mathbf{m}_i \in \mathbb{R}^T$ using the price vector $\mathbf{q}$ in the USER(i) problem

  The cloud service provider receives the budget matrix $\mathbf{m}$ and does the following

  step = 0

  while *step* $\leq G$ **do**

    $q_t = \max\left[q_t + \kappa\left(\frac{\sum_{k=1}^{K}\sum_{i=1}^{N} m_{it}^k \delta_k}{q_t} - M_t\right), 0\right] \ \forall\, t \in [T]$

    $step = step + 1$

  end

end

$z_{it}^k = m_{it}^k / q_t$ for all $i \in [N], t \in [T]$

Use Algorithm 22 to project z_{it}^k to the constraint sets $\sum_k \sum_i z_{it}^k \leq M_t, \ \forall\, t$ and $z_{it}^k \geq 0 \ \forall\, i, t, k$.

Result: Solution to SYS, that is, the optimal pricing vector $\mathbf{q}^*$ and the optimal resource allocation vector $\mathbf{x}^*$

number of users is 20, we keep the demand and supply of the system almost the same. In that case, the CPT system utility is only 4% better than the EU utility. However, when the number of users is increased to 100 while keeping the supply constant, the CPT system utility is 20% better. For the rest of the experiments in this section, we consider the latter case (i.e., when the demand exceeds supply by approximately a factor of five).

In Fig. 10.6, we show the effect of having CPT users in the system and how does that affect the overall system utility. Since the EU problem does the account for the CPT behaviour, the system utility does not change much. Further, we note that the cloud provider can significantly increase the system utility if it takes into account the CPT behaviour of the users.

10.5.1 Market Simulations

In this section, we run an experiment for 60 days with $N = 100$ users and $T = 3$ tiers, where each day, users' utilities are changing based on the market trends. Specifically, for the first 30 days, we synthetically generate utilities that follow an upward trend based on the market following by a downward trend for the next 30 days. To simulate this, we generate u_{it}, for all i, t, from a uniform distribution in $[5, 10]$ in an i.i.d. fashion. The job size for the i-th user, J_i, for all i, is an i.i.d. integer

Algorithm 22: Han's algorithm for projection on the intersection of convex sets

Input : $\mathbf{z} \in \mathbb{R}^{K \times N \times T}$, Constraint sets $C_1, C_2 \in \mathbb{R}^{K \times N \times T}$, where $C_1 := \{z_{it}^k | \ \sum_k \sum_i z_{it}^k \leq M_t, \ \forall \ t\}$, and $C_2 := \{z_{it}^k | \ z_{it}^k \geq 0 \ \forall \ i, t, k\}$, error tolerance ϵ

Initialization: Define $\mathbf{z}_1 = \mathbf{z}, \quad \mathbf{z}_2 = \mathbf{z}$.

while $\|\mathbf{z} - \mathbf{z}_{prev}\| \geq \epsilon \|\mathbf{z}_{prev}\|$ **do**

  $\mathbf{z}_{prev} = \mathbf{z}$

  $(z_1')_{it}^k = z_{it}^k - \delta_k \max \left[\frac{\sum_{i,k} z_{it}^k \delta_k - M_t}{N \sum \delta_k^2}, 0 \right] \ \forall \ i, t, k.$

  $\mathbf{z}_2' = \mathbf{z}, \quad \mathbf{z}_2'[\mathbf{z}_2' < 0] = 0$

  $\mathbf{z} = (\mathbf{z}_1' + \mathbf{z}_2')/2$

  $\mathbf{z}_1 = \mathbf{z} + \mathbf{z}_1 - \mathbf{z}_1'$

  $\mathbf{z}_2 = \mathbf{z} + \mathbf{z}_2 - \mathbf{z}_2'$

end

Result: $\mathbf{z}$: Projection of the input matrix to sets C_1 and C_2

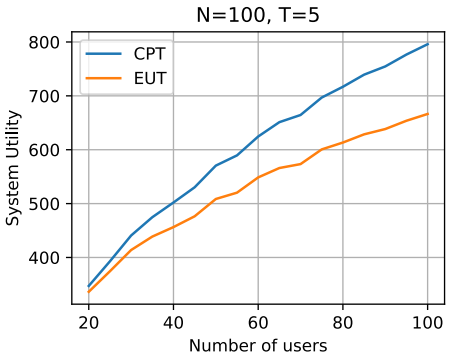


Figure 10.5: CPT versus EU Utilities when the number of users (that is, demand) is increased.

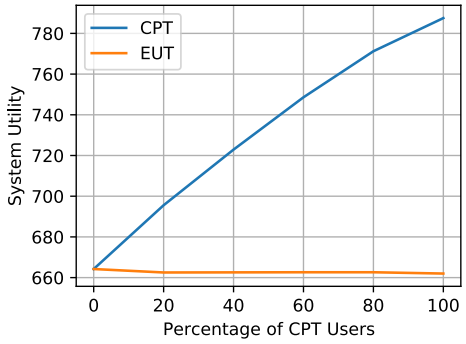
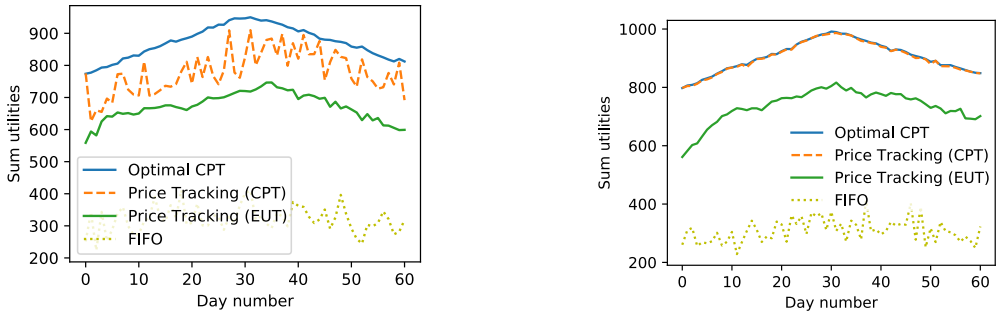


Figure 10.6: CPT versus EU Utilities when the percentage of users that exhibit CPT behavior is increased.

chosen between 10 to 100 (which remains constant throughout the 60 days of the experiment) and $M_t = 5000$ for all t. To generate an upward market trend, we add 0.5 to $u_{it}, \forall i, t$, with probability 0.55 and -0.5 with probability 0.45 (the probabilities are flipped to generate a downward trend).

We compare the following four schemes:

- **Optimal CPT pricing**: Here, we assume that the cloud provider is able to solve for the optimal lotteries and maximize system utility by utilizing the knowledge of users' utilities at each day.
- **Price Tracking (CPT)**: In this case, the users' utilities are not known, and the cloud provider tracks users' utilities based on their budget signals (as described in Algorithm 21). The cloud provider is assumed to update the prices everyday. *Users send budget signals only once per*



(a) One user budget update each day for the price tracking schemes (EU and CPT)

(b) Two user budget updates each day for the price tracking algorithms (EU and CPT)

Figure 10.7: Sum utilities for the four resource allocation schemes

day. These budget signals depend on the user's utility function on that day and the prices published by the cloud on that day.

- **Price Tracking (EUT)**: Here, we do not consider the CPT-ness of users and use the price tracking algorithm from earlier work.
- **First-In-First-Out (FIFO)**: Here, the pricing of the resources remains constant (as currently employed by most commercial service providers). It is the optimum prices determined by the utilities on day 1 and does not capture the mood of the market. Each day, the users are allocated resources on a first-come-first-serve (which is assumed to be random in order) or a lottery basis.

In Fig. 10.7, we compare the four schemes in terms of overall system utility. Fig. 10.7a shows the case where the tracking algorithms are limited to only one interaction between the user and the cloud. The best hyperparameters (obtained through tuning) were chosen for both CPT and EUT tracking pricing schemes (step-size and the number of gradient updates). The CPT-based tracking scheme from Algorithm 21 is always within 25% of the optimal system utility.

Further, when we increase the number of interactions between users and the cloud provider to two (see Fig. 10.7b), we observe that the CPT price tracking algorithms emulates the optimal CPT solution (which is obtained through the knowledge of user utility at the cloud). Further, in both cases, the price tracking with the EU solution and first-come-first-serve are suboptimal. For the rest of the experiments in this section, we consider the case case when the user and the cloud are interacting only once per day.

In Fig. 10.8, we focus on one low-utility user in the high-demand-low-supply system. We identify the low-utility user by plotting its normalized utility across tiers (see Fig. 10.8d), where the normalized utility for the i-th user in the t-th tier is given by is defined as

$$\frac{U_i(t)/J_i}{\sum_i U_i(t)/\sum_i J_i}.$$

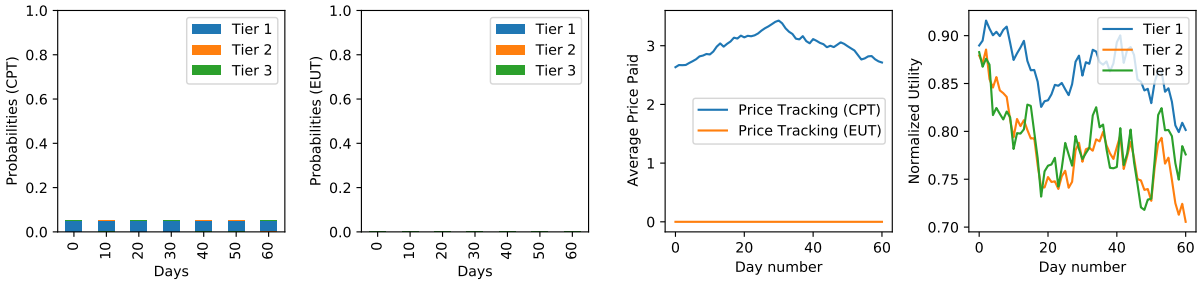


Figure 10.8: Resource allocation statistics for a low-utility user

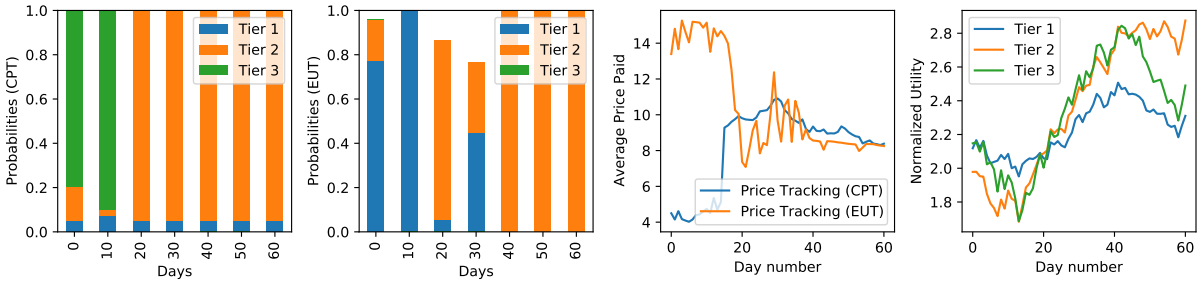


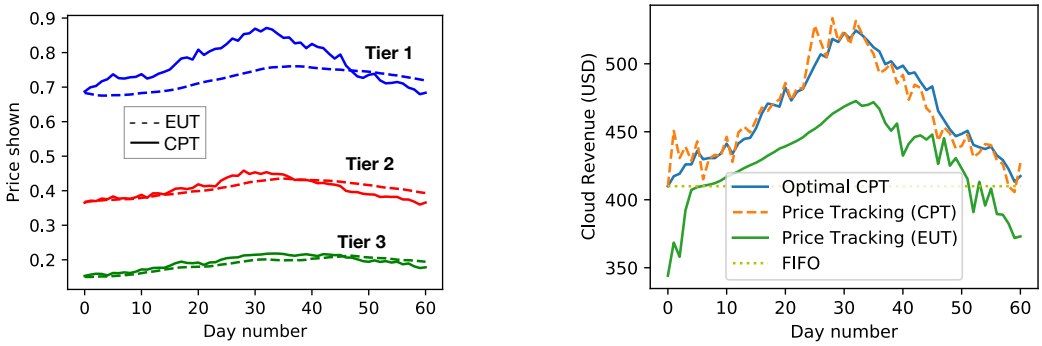
Figure 10.9: Resource allocation statistics for a high-utility user

We see that the normalized utility of the given user is much less than the average utility per job of the system.

In Fig. 10.8(a) and (b), we plot the bar graph of probabilities that the user gets all her jobs allocated for the three tiers in the CPT and EUT cases, respectively. This is done for days numbered $0, 10, 20, 30, 40, 50, 60$. As expected, in the EUT case, the user does not get any resources allocated across the 60 days of the simulation due to her low utility. However, the CPT allocation assigns a small probability of allocating resources to the user, taking advantage of the positive weighing assigned to small probabilities in the CPT function (Fig. 10.2). Thus, when we plot the expected revenue obtained from the user by the cloud in Fig. 10.8(c), we see that the CPT objective (exact and tracking-based) is able to extract a non-zero revenue on average from the low-utility user by assigning a non-zero chance of giving her resources in the lottery.

In Fig. 10.9, we focus on a high-utility user. As can be seen from Fig. 10.9(d), the user has a higher-than-average normalized utility. We note from Fig. 10.9(a) that the user is getting resources in the first tier with a small probability in the CPT case, again utilizing the positive weighing of the CPT function for smaller lotteries. However, in the EUT case as well, the user is getting resources allocated with high probability since the user has a high utility, but notice that in days 0, 20 and 30, the user does not get resources allocated with probability one (across all the three tiers). This does not happen in the CPT case since it under-weighs high probabilities that are less than one.

In Fig. 10.10, we show the effect of different pricing mechanisms on the prices charged by the cloud and its corresponding revenue. First-come-first-serve has a constant pricing independent of the market and the users' utilities, similar to the setup in present day pricing employed by the cloud.



(a) Price Charged by the Cloud in the first tier (b) Expected revenue of the Cloud

Figure 10.10: Price tracking using only one user-budget update per day is close to optimal for the CPT case. Further, the average revenue of the Cloud increased with the CPT allocation while maximizing the system utility.

In Fig. 10.10a, we plot the prices charged by the cloud provider in the first tier, and observe that in the CPT case the price follows the trend of the market, while the EU price tracking is suboptimal.

In Fig. 10.10b, we plot the total expected revenue of the cloud for the four schemes. CPT-based schemes, by the virtue of assigning lotteries to low-utility users, is able to perform better than EU based scheme that does not take into account the CPT weighings. As expected, the revenue obtained by the first-come-first-serve scheme is constant. Even though it is better than the EU based scheme during some days, assigning resources to users randomly decreases the overall utility of the system (Fig. 10.7).

10.6 Proofs

10.6.1 Proof of Proposition 14

We will decompose the problem SYS-CPT-K-R into user problems and a central problem in order to decouple the different users. The decomposition here is different from the one given in section 10.4. For each user i, consider the follwoing user problem:

USER (i)-v2

$$\max_{z_{i,t}(k)\geq 0} \quad \sum_{k=1}^{K} h_i(k)\bar{F}_i(z_{i,\bullet}(k)) - \frac{1}{K}\sum_{k=1}^{K}\sum_{t=1}^{T}\bar{\rho}_t z_{i,t}(k)$$

$$\text{subject to} \quad \sum_{t=1}^{T} z_{i,t}(k) \leq J_i, \ \forall\, k \in [K], \tag{10.74}$$

$$\bar{F}_i(z_{i,\bullet}(k)) \geq \bar{F}_i(z_{i,\bullet}(k+1)), \ \forall\, 1 \leq k \leq K-1. \tag{10.75}$$

Let the central problem we given by

CENTRAL

$$\begin{aligned} \max_{z_{i,t}^k \geq 0} \quad & \sum_{i=1}^{N} \frac{1}{K} \sum_{k=1}^{K} \sum_{t=1}^{T} \bar{\rho}_t z_{i,t}(k) \\ \text{subject to} \quad & \sum_{i=1}^{N} \frac{1}{K} \sum_{k=1}^{K} z_{i,t}(k) \leq M_t, \ \forall \, t \in [T]. \end{aligned} \tag{10.76}$$

We will first show that there exists equilibrium prices $\bar{\rho}_t$ such that any solution $(\mathbf{z}(k))_k$ to SYS-CPT-K-R is a common solution to USER(i), for all i, and CENTRAL problems, and vice versa, i.e. a common solution to to USER(i), for all i, and CENTRAL problems is a solution to SYS-CPT-K-R. This essentially follows from showing that the complementary slackness conditions match.

The Lagrangian for the user problem USER(i)-v2 is given by

$$\begin{aligned} L((\mathbf{z}(k)_k), (\lambda_i(k))_{i,k}, (\alpha_i(k)_{i,k})) = & \sum_{k=1}^{K} h_i(k) \bar{F}_i(z_{i,\bullet}(k)) - \frac{1}{K} \sum_{k=1}^{K} \sum_{t=1}^{T} \bar{\rho}_{i,t} z_{i,t}(k) \\ & - \sum_{i=1}^{N} \sum_{k=1}^{K} \lambda_i(k) \left(J_i - \sum_{t=1}^{T} z_{i,t}(k) \right) \\ & - \sum_{i=1}^{N} \sum_{k=1}^{K} \alpha_i(k) \left(\sum_{t=1}^{T} F_{i,t} z_{i,t}(k) - \sum_{t=1}^{T} F_{i,t} z_{i,t}(k+1) \right), \end{aligned}$$

where $\lambda_i(k) \geq 0$ and $\alpha_i(k) \geq 0$ are the dual variables corresponding to constraints (10.74) and (10.75). The complementary slackness conditions for the USER(i)-v2 problem are:

$$h_i(k) F_{i,t} - \lambda_i(k) - \frac{\rho_{i,t}}{k} + F_{i,t}(\alpha_i(k) - \alpha_i(k-1)) \begin{cases} = 0, & \text{if } z_{i,t}(k) > 0, \\ \geq 0, & \text{if } z_{i,t}(k) = 0, \end{cases} \qquad \forall i, t, k, \tag{10.77}$$

$$\sum_{t=1}^{T} z_{i,t}(k) \begin{cases} = J_i, & \text{if } \lambda_i(k) > 0, \\ \leq J_i, & \text{if } \lambda_i(k) = 0, \end{cases} \qquad \forall i, k, \tag{10.78}$$

$$\sum_{t=1}^{T} F_{i,t} z_{i,t}(k) - \sum_{t=1}^{T} F_{i,t} z_{i,t}(k+1) \begin{cases} = 0, & \text{if } \alpha_i(k) > 0, \\ \geq 0, & \text{if } \alpha_i(k) = 0, \end{cases} \qquad \forall i, k. \tag{10.79}$$

The Lagrangian for the CENTRAL problem is given by

$$L((\mathbf{z}(k)_k), (\mu_t)_t) = \sum_{i=1}^{N} \frac{1}{K} \sum_{k=1}^{K} \sum_{t=1}^{T} \bar{\rho}_t z_{i,t}(k) - \sum_{t=1}^{T} \mu_t \left(M_t - \sum_{i=1}^{N} \frac{1}{K} \sum_{k=1}^{K} z_{i,t}(k) \right),$$

where $\mu_t \geq 0$ are the dual variables corresponding to constraint (10.76). The complementary slackness conditions are given by

$$\frac{1}{K}\bar{\rho}_t - \frac{1}{K}\mu_t \begin{cases} = 0, & \text{if } z_{i,t}(k) > 0, \\ \geq 0, & \text{if } z_{i,t}(k) = 0, \end{cases} \qquad \forall i, t, k \tag{10.80}$$

$$\frac{1}{K}\sum_{i=1}^{N}\sum_{k=1}^{K} z_{i,t}(k) \begin{cases} = M_t, & \text{if } \mu_t > 0, \\ \leq M_t, & \text{if } \mu_t = 0, \end{cases} \qquad \forall t, \tag{10.81}$$

Now, notice that if we have an optimal solution to SYS-CPT-K-R along with its dual variables then taking the equilibrium prices $\bar{\rho}_t = \mu_t, \forall t$, we get that these variables also satisfy the complementary slackness conditions for all user problems USER(i)-v2 and the CENTRAL problem. On the other hand, if we have equilirbrium prices $\bar{\rho}_t$ for which we have a solution to the user problems USER(i)-v2 and the CENTRAL problem with a common solution $(\mathbf{z}(k))_k$, then it follows that the corresponding dual variables also satisfy the complementary slackness conditions for SYS-CPT-K-R. This shows that the converse is also true.

This decomposition allows us to decouple the preferences of the different users. Note that the preference features of each player, namely, her utility and probability weighting functions, appear in her corresponding USER problem. And the CENTRAL problem correlates different players' preferences via psuedo prices $\bar{\rho}_t$. We will use this decomposition to study the typical lottery allocation received by a player by analysing the USER(i)-v2 problem.

For $Z_i \geq 0$, let us define

$$\begin{aligned} W_i(Z_i) := \max_{\bar{z}_{i,t} \geq 0} \quad & \sum_{t=1}^{T} F_{i,t}\bar{z}_{i,t} \\ \text{subject to} \quad & \sum_{t=1}^{T} \bar{\rho}_t \bar{z}_{i,t} \leq Z_i, \\ & \sum_{t=1}^{T} \bar{z}_{i,t} \leq J_i. \end{aligned}$$

We observe that the function $W_i(Z_i)$ is non-decreasing, continuous, differentiable, and concave in z_i. The USER(i)-v2 problem can now we written as

USER (i)-v3

$$\begin{aligned} \max_{Z_i(k) \geq 0} \quad & \sum_{k=1}^{K} h_i(k) W_i(Z_i(k)) - \frac{1}{K}\sum_{k=1}^{K} Z_i(k) \\ \text{subject to} \quad & W_i(Z_i(k)) \geq W_i(Z_i(k+1)), \ \forall\, 1 \leq k \leq K, \end{aligned} \tag{10.82}$$

where $Z_i(K+1) = 0$.

It is proved in [248] that for a non-decreasing concave function, any optimal solution $(Z_i^*(k))_k$ for USER(i)-v3 satisfies

$$W_i(Z_i^*(k^*)) = W_i(Z_i^*(k^*+1)) = \cdots = W_i(Z_i^*(K)),$$

where $k^* = \min\{k \in [K] : (k-1)/K \geq p_i^*\}$, provided $p_i^* \leq (K-1)/K$.[1] This proves proposition 3.

10.6.2 Proof of Theorem 15

Let $\mathbf{z}^R$ be an optimal resource allocation for SYS-CPT-K-R. Suppose $(k_1, k_2, i) \in \mathcal{T}$ corresponding to $\mathbf{z}^R$. That is, user i is getting non-zero resources in slots m and n (and say $m < n$) in alternatives k_1 through k_2, i.e. $z_{im}(k) > 0, x_{in}(k) > 0, \forall k_1 \leq k \leq k_2$. Recall that $z_{i,\bullet}^R(k)$ are identical for all $k_1 \leq k \leq k_2$. Consider a user j which is getting positive resources in slot m and alternative k', i.e. $z_{jm}(k') > 0$.

Suppose we redistribute $\epsilon(>0)$ fraction of the job from user j in slot m and alternative k' to user i in slot m and alternatives $k \in [k_1, k_2]$ (equally distributed) and vice versa in slot n. By the optimality of $\mathbf{z}^R$, we know that this can only decrease the objective function in SYS-CPT-K-R, that is

$$\left(\frac{\sum_{k=k_1}^{k_2} h_i(k)}{k_2 - k_1 + 1}\right)(F_{im} - F_{in})\epsilon - h_j(k)(F_{jm} - F_{jn})\epsilon \leq 0.$$

Then, , we have

$$\left(\frac{\sum_{k=k_1}^{k_2} h_i(k)}{k_2 - k_1 + 1}\right)(F_{im} - F_{in}) \leq h_j(k')(F_{jm} - F_{jn}). \tag{10.83}$$

Now, we are ready to prove the theorem. From Eq. (10.48), we get that

$$\frac{V^R - V^*}{V^R} \leq \frac{V^R - \tilde{V}}{V^R}$$

where $\tilde{V}$ is the objective of SYS-CPT-K at $\tilde{\mathbf{z}}$ obtained by projecting solution of SYS-CPT-K-R as described in Eq. (10.42), (10.43), and (10.44).

Now, our aim is to upper bound the RHS above to get an upper bound on the gap. To that end, we bound the numerator and denominator at each time slot $t \in [T]$. Let V_t^R and $\tilde{V}_t$ be the corresponding utilities obtained only at tier $t \in [T]$, i.e.

$$V_t^R := \sum_{i=1}^{N}\sum_{k=1}^{K} h_i(k) F_{it} z_{it}(k) \quad \text{and} \quad \tilde{V}_t := \sum_{i=1}^{N}\sum_{k=1}^{K} h_i(k) F_{it} \tilde{z}_{it}(k).$$

[1] In [248], it is assumed that the function $W_i(\cdot)$ is strictly increasing, continuous, differentiable and strictly concave. However, the same proof allows us to show the required property under the weaker conditions that the function $W_i(\cdot)$ is non-decreasing, continuous, differentiable, and concave. In [248], the authors show a stronger result where

$$Z_i^*(k^*) = Z_i^*(k^*+1) = \cdots = Z_i^*(K),$$

for which they need the function $W_i(\cdot)$ to be strictly increasing.

Let $\tilde{T}_i$ be the time at which the job of user i are getting finished according to the resource allocation $\mathbf{z}^R$ as defined in (10.42).

Thus, for time slot m, we have

$$\frac{V_m^R - \tilde{V}_m}{V_m^R} = \frac{\sum_{(k_1,k_2;i)\in\mathcal{T}_m} \sum_{k=k_1}^{k_2} h_i(k)(F_{im} - F_{i\tilde{T}_i^k}) z_{im}^R(k)}{\sum_{j=1}^N \sum_{k=1}^K F_{jm} z_{jm}^R(k)}, \tag{10.84}$$

where $\mathcal{T}_m$ is the set of elements $(k_1, k_2; i) \in \mathcal{T}$ for which $0 < z_{im}^R(k) < J_i$, for $k_1 \le k \le k_2$. We can further write

$$\frac{V_m^R - \tilde{V}_m}{V_m^R} \le \sum_{(k_1,k_2;i)\in\mathcal{T}_m} \frac{(\sum_{k=k_1}^{k_2} h_i(k))(F_{im} - F_{i\tilde{T}_i^k}) z_{it}^R(k)}{\sum_{j=1}^N \sum_{k=1}^K h_j(k)(F_{jm} - F_{j\tilde{T}_i^k}) z_{jt}^R(k)} \tag{10.85}$$

$$\le \sum_{(k_1,k_2;i)\in\mathcal{T}_m} \frac{(\sum_{k=k_1}^{k_2} h_i(k))(F_{im} - F_{i\tilde{T}_i^k}) z_{it}^R(k)}{\sum_{j=1}^N \sum_{k=1}^K \frac{\sum_{k=k_1}^{k_2} h_i(k)}{k_2 - k_1 + 1}(F_{i,m} - F_{i,\tilde{T}_i^k}) z_{jt}^R(k)}, \tag{10.86}$$

where the last inequality uses Eq. (10.83). Now, since at time m, user i is getting fractional resources, it implies that the system is operating at full capacity, that is

$$\frac{1}{K} \sum_{j=1}^N \sum_{k=1}^K z_{jt}^R(k) = M_t.$$

Hence, we get

$$\frac{V_t^R - \tilde{V}_t}{V_t^R} \le \sum_{(k_1,k_2;i)\in\mathcal{T}_m} \frac{K z_{it}^R(k)}{(k_2 - k_1 + 1) M_t} \le \sum_{(k_1,k_2;i)\in\mathcal{T}_m} \frac{\max_i(J_i)}{M_t}, \tag{10.87}$$

where the last inequality uses the fact that $z_{it}^R(k) \le \max_i(J_i)$ and $k_2 - k_1 + 1 \le K$. Also, since $\mathcal{T} \le T$, we get

$$\frac{V_t^R - \tilde{V}_t}{V_t^R} \le \frac{T \max_i(J_i)}{M_t} \le \frac{T \max_i(J_i)}{\min_t M_t}. \tag{10.88}$$

Thus,

$$V^R - V^* \le V^R - \tilde{V} = \sum_{t=1}^T (V_t^R - \tilde{V}_t) \le \frac{T \max_i(J_i)}{\min_t M_t} \sum_{t=1}^T V_t^R = \frac{T \max_i(J_i)}{\min_t M_t} V^R, \tag{10.89}$$

Rearranging, we get

$$V^* \ge \left(1 - \frac{T(\max_i J_i)}{\min_t M_t}\right) V^R,$$

which proves the desired result.

Bibliography

[1] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%", in *Proceedings of the 2017 Symposium on Cloud Computing*, ACM, 2017, pp. 445–451.

[2] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, "Serverless computing: Current trends and open problems", in *Research Advances in Cloud Computing*, S. Chaudhary, G. Somani, and R. Buyya, Eds. Springer Singapore, 2017, vol. abs/1706.03178.

[3] V. Shankar, K. Krauth, Q. Pu, E. Jonas, S. Venkataraman, I. Stoica, B. Recht, and J. Ragan-Kelley, "Numpywren: Serverless Linear Algebra", *ArXiv e-prints*, vol. abs/1810.09679, Oct. 2018. arXiv: `1810.09679 [cs.DC]`. [Online]. Available: `http://arxiv.org/abs/1810.09679`.

[4] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back", *arXiv preprint arXiv:1812.03651*, 2018.

[5] V. Gupta, S. Wang, T. Courtade, and K. Ramchandran, "Oversketch: Approximate matrix multiplication for the cloud", in *2018 IEEE International Conference on Big Data (Big Data)*, Dec. 2018, pp. 298–304.

[6] V. Gupta, S. Kadhe, T. Courtade, M. W. Mahoney, and K. Ramchandran, "Oversketched newton: Fast convex optimization for serverless systems", in *2020 IEEE International Conference on Big Data (Big Data)*, IEEE, 2020, pp. 288–297.

[7] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, *et al.*, "Cloud programming simplified: A berkeley view on serverless computing", *arXiv preprint arXiv:1902.03383*, 2019.

[8] J. Spillner, C. Mateos, and D. A. Monge, "Faaster, better, cheaper: The prospect of serverless scientific computing and HPC", in *Latin American High Performance Computing Conference*, Springer, 2017, pp. 154–168.

[9] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, J. Carreira, N. J. Yadwadkar, R. A. Popa, J. E. Gonzalez, I. Stoica, and D. A. Patterson, "What serverless computing is and should become: The next phase of cloud computing", *Communications of the ACM*, vol. 64, no. 5, pp. 76–84, 2021.

[10] S. Jhakotia, *Serverless architecture market by end-users and geography - global forecast 2019-2023*, https://medium.com/@suryaj/why-serverless-is-the-future-of-cloud-computing-45e417dc4018, 2018.

[11] Technavio, *Serverless architecture market by end-users and geography - global forecast 2019-2023*, https://www.technavio.com/report/serverless-architecture-market-industry-analysis.

[12] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform", in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, 2018, pp. 257–262.

[13] A. Aytekin and M. Johansson, "Harnessing the power of serverless runtimes for large-scale optimization", *arXiv preprint arXiv:1901.03161*, 2019.

[14] H. Wang, D. Niu, and B. Li, "Distributed machine learning with a serverless architecture", in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 1288–1296.

[15] L. Feng, P. Kudva, D. Da Silva, and J. Hu, "Exploring serverless computing for neural network training", in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, Jul. 2018, pp. 334–341.

[16] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz, "Cirrus: A serverless framework for end-to-end ml workflows", in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 13–24.

[17] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Communication-optimal parallel and sequential cholesky decomposition", *SIAM Journal on Scientific Computing*, vol. 32, no. 6, pp. 3495–3523, 2010.

[18] J. Dean and L. A. Barroso, "The tail at scale", *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[19] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the influence of system noise on large-scale applications by simulation", in *Proc. of the ACM/IEEE Int. Conf. for High Perf. Comp., Networking, Storage and Analysis*, 2010, pp. 1–11.

[20] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, fast and slow: Low-latency video processing using thousands of tiny threads", in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 363–376.

[21] L. Feng, P. Kudva, D. Da Silva, and J. Hu, "Exploring serverless computing for neural network training", in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, 2018, pp. 334–341.

[22] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[23] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets", in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, Boston, MA, 2010, pp. 10–10.

[24] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes", *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, Dec. 2018.

[25] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning", in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, PMLR, 2017, pp. 3368–3376.

[26] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication", in *IEEE Int. Sym. on Information Theory (ISIT), 2017*, IEEE, 2017, pp. 2418–2422.

[27] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes", in *IEEE Int. Sym. on Information Theory (ISIT), 2018*, IEEE, 2018, pp. 1993–1997.

[28] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication", in *Adv. in Neural Inf. Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 4403–4413.

[29] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication", *IEEE Transactions on Information Theory*, vol. 66, no. 1, pp. 278–301, 2019.

[30] B. Bartan and M. Pilanci, "Polar coded distributed matrix multiplication", *arXiv preprint arXiv:1901.06811*, 2019.

[31] H. Jeong, F. Ye, and P. Grover, "Locally recoverable coded matrix multiplication", in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2018, pp. 715–722.

[32] H. Jeong, Y. Yang, V. Gupta, C. Engelmann, T. M. Low, V. Cadambe, K. Ramchandran, and P. Grover, "3d coded summa: Communication-efficient and robust parallel matrix multiplication", in *European Conference on Parallel Processing*, Springer, 2020, pp. 392–407.

[33] A. M. Subramaniam, A. Heidarzadeh, and K. R. Narayanan, "Collaborative decoding of polynomial codes for distributed computation", in *2019 IEEE Information Theory Workshop (ITW)*, IEEE, 2019, pp. 1–5.

[34] J. Zhu, Y. Pu, V. Gupta, C. Tomlin, and K. Ramchandran, "A sequential approximation framework for coded distributed optimization", in *Annual Allerton Conf. on Communication, Control, and Computing, 2017*, IEEE, 2017, pp. 1240–1247.

[35] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products", in *Advances In Neural Information Processing Systems*, 2016, pp. 2100–2108.

[36] J. Kosaian, K. Rashmi, and S. Venkataraman, "Learning-based coded computation", *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 227–236, 2020.

[37] Y. Yang, M. Chaudhari, P. Grover, and S. Kar, "Coded iterative computing using substitute decoding", *arXiv preprint arXiv:1805.06046*, 2018.

[38] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems", in *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 2017, pp. 709–719.

[39] M. Ye and E. Abbe, "Communication-computation efficient gradient coding", in *International Conference on Machine Learning*, PMLR, 2018, pp. 5610–5619.

[40] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast monte carlo algorithms for matrices I: Approximating matrix multiplication", *SIAM Journal on Computing*, vol. 36, no. 1, pp. 132–157, 2006.

[41] D. P. Woodruff, "Sketching as a tool for numerical linear algebra", *Found. Trends Theor. Comput. Sci.*, vol. 10, no. 1–2, pp. 1–157, 2014.

[42] M. Pilanci and M. J. Wainwright, "Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence", *SIAM Jour. on Opt.*, vol. 27, no. 1, pp. 205–245, 2017.

[43] Y. Yang, M. Pilanci, and M. J. Wainwright, "Randomized sketches for kernels: Fast and optimal non-parametric regression", *stat*, vol. 1050, pp. 1–25, 2015.

[44] F. Roosta-Khorasani and M. W. Mahoney, "Sub-Sampled Newton Methods I: Globally Convergent Algorithms", *arXiv e-prints*, arXiv:1601.04737, arXiv:1601.04737, Jan. 2016. arXiv: 1601.04737 [math.OC].

[45] ——, "Sub-Sampled Newton Methods II: Local Convergence Rates", *arXiv e-prints*, arXiv:1601.04738, arXiv:1601.04738, Jan. 2016. arXiv: 1601.04738 [math.OC].

[46] P. Xu, F. Roosta, and M. W. Mahoney, "Newton-type methods for non-convex optimization under inexact hessian information", 1, vol. 184, Springer, 2017, pp. 35–70. [Online]. Available: https://arxiv.org/abs/1708.07164.

[47] M. W. Mahoney, *Randomized algorithms for matrices and data*, ser. Foundations and Trends in Machine Learning 2. Boston: NOW Publishers, 2011, vol. 3, pp. 123–224.

[48] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate Newton-type method", in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14, Beijing, China: JMLR.org, 2014, pp. II-1000–II-1008.

[49] Y. Zhang and X. Lin, "Disco: Distributed optimization for self-concordant empirical loss", in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, Jul. 2015, pp. 362–370.

[50] S. J. Reddi, A. Hefny, S. Sra, B. Pöczos, and A. Smola, "On variance reduction in stochastic gradient descent and its asynchronous variants", in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 2647–2655.

[51] S. Wang, F. Roosta-Khorasani, P. Xu, and M. W. Mahoney, "GIANT: Globally improved approximate Newton method for distributed optimization", in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2018, pp. 2332–2342.

[52] C. Duenner, A. Lucchi, M. Gargiani, A. Bian, T. Hofmann, and M. Jaggi, "A distributed second-order algorithm you can trust", in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 1358–1366.

[53] V. Smith, S. Forte, C. Ma, M. Takác, M. I. Jordan, and M. Jaggi, "Cocoa: A general framework for communication-efficient distributed optimization", *arXiv preprint arXiv:1611.02189*, 2016.

[54] J. Konecny, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency", *Preprint arXiv:1610.05492*, 2016.

[55] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data", in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, vol. 54, 2017, pp. 1273–1282.

[56] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning", *arXiv preprint arXiv:1912.04977*, 2019.

[57] S. U. Stich, "Local SGD converges fast and communicates little", in *International Conference on Learning Representations*, 2019. [Online]. Available: `https://openreview.net/forum?id=S1g2JnRcFX`.

[58] F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. Cadambe, "Local SGD with periodic averaging: Tighter analysis and adaptive synchronization", in *Advances in Neural Information Processing Systems*, 2019, pp. 11 080–11 092.

[59] A. Dieuleveut and K. K. Patel, "Communication trade-offs for local-sgd with large step size", in *Advances in Neural Information Processing Systems*, 2019, pp. 13 579–13 590.

[60] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi, "Don't Use Large Mini-batches, Use Local SGD", in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1eyO1BFPr.

[61] V. Gupta, S. A. Serrano, and D. DeCoste, "Stochastic weight averaging in parallel: Large-batch training that generalizes well", in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rygFWAEFwS.

[62] M. Dereziński and M. W. Mahoney, "Distributed estimation of the inverse hessian by determinantal averaging", in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 11 405–11 415.

[63] M. Dereziński, B. Bartan, M. Pilanci, and M. W. Mahoney, "Debiasing distributed second order optimization with surrogate sketching and scaled regularization", *arXiv preprint arXiv:2007.01327*, 2020.

[64] A. Ghosh, R. K. Maity, and A. Mazumdar, "Distributed newton can communicate less and resist byzantine workers", in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 18 028–18 038. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/d17e6bcbcef8de3f7a00195cfa5706f1-Paper.pdf.

[65] A. Ghosh, R. K. Maity, A. Mazumdar, and K. Ramchandran, "Communication efficient distributed approximate newton method", in *2020 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 2539–2544.

[66] Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney, "Pyhessian: Neural networks through the lens of the hessian", in *2020 IEEE International Conference on Big Data (Big Data)*, IEEE, 2020, pp. 581–590.

[67] Z. Yao, A. Gholami, S. Shen, K. Keutzer, and M. W. Mahoney, "Adahessian: An adaptive second order optimizer for machine learning", *arXiv preprint arXiv:2006.00719*, 2020.

[68] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning", in *Proc. of the 26th Annual International Conference on Machine Learning*, ACM, Montreal, Quebec, Canada: ACM, 2009, pp. 1113–1120.

[69] A. Aghazadeh, R. Spring, D. Lejeune, G. Dasarathy, A. Shrivastava, *et al.*, "Mission: Ultra large-scale feature selection using count-sketches", in *International Conference on Machine Learning*, PMLR, 2018, pp. 80–88.

[70] K. S. Tai, V. Sharan, P. Bailis, and G. Valiant, "Sketching Linear Classifiers over Data Streams", in *Proc. of the 2018 Intl. Conf. on Management of Data*, ACM, 2018, pp. 757–772.

[71] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams", in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ser. ICALP '02, Springer, Berlin, Heidelberg: Springer-Verlag, 2002, pp. 693–703. [Online]. Available: http://dl.acm.org/citation.cfm?id=646255.684566.

[72] S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team, "An empirical model of large-batch training", *arXiv preprint arXiv:1812.06162*, 2018.

[73] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima", *arXiv preprint arXiv:1609.04836*, 2016.

[74] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: Closing the generalization gap in large batch training of neural networks", in *NIPS*, 2017.

[75] N. Golmant, N. Vemuri, Z. Yao, V. Feinberg, A. Gholami, K. Rothauge, M. W. Mahoney, and J. Gonzalez, "On the computational inefficiency of large batch sizes for stochastic gradient descent", *arXiv preprint arXiv:1811.12941*, 2018.

[76] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training", *arXiv preprint arXiv:1811.03600*, 2018.

[77] A. Devarakonda, M. Naumov, and M. Garland, "Adabatch: Adaptive batch sizes for training deep neural networks", *CoRR*, vol. abs/1712.02029, 2017. arXiv: 1712.02029. [Online]. Available: http://arxiv.org/abs/1712.02029.

[78] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour", *arXiv preprint arXiv:1706.02677*, 2017.

[79] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, *et al.*, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes", *arXiv preprint arXiv:1807.11205*, 2018.

[80] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size", *arXiv preprint arXiv:1711.00489*, 2017.

[81] Y. You, I. Gitman, and B. Ginsburg, "Scaling SGD batch-size to 32k for imagenet training", *arXiv preprint arXiv:1708.03888*, 2017.

[82] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, "Averaging weights leads to wider optima and better generalization", *arXiv preprint arXiv:1803.05407*, 2018.

[83] E. Nikishin, P. Izmailov, B. Athiwaratkun, D. Podoprikhin, T. Garipov, P. Shvechikov, D. Vetrov, and A. G. Wilson, "Improving stability in deep reinforcement learning with weight averaging".

[84] B. Athiwaratkun, M. Finzi, P. Izmailov, and A. G. Wilson, "There are many consistent explanations of unlabeled data: Why you should average", in *International Conference on Learning Representations*, 2019. [Online]. Available: `https://openreview.net/forum?id=rkgKBhA5Y7`.

[85] W. Maddox, T. Garipov, P. Izmailov, D. Vetrov, and A. G. Wilson, "A simple baseline for bayesian uncertainty in deep learning", *arXiv preprint arXiv:1902.02476*, 2019.

[86] G. Yang, T. Zhang, P. Kirichenko, J. Bai, A. G. Wilson, and C. De Sa, "SWALP : Stochastic weight averaging in low precision training", in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, Long Beach, California, USA: PMLR, Sep. 2019, pp. 7015–7024.

[87] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", *arXiv preprint arXiv:1810.04805*, 2018.

[88] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis, "Compressed Communication for Distributed Deep Learning: Survey and Quantitative Evaluation", Tech. Rep., 2020.

[89] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding", in *Advances in Neural Information Processing Systems*, 2017, pp. 1709–1720.

[90] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signSGD: Compressed Optimisation for Non-Convex Problems", in *International Conference on Machine Learning*, 2018, pp. 560–569.

[91] S. P. Karimireddy, Q. Rebjock, S. Stich, and M. Jaggi, "Error Feedback Fixes SignSGD and other Gradient Compression Schemes", in *International Conference on Machine Learning*, 2019, pp. 3252–3261.

[92] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning", in *Advances in neural information processing systems*, 2017, pp. 1509–1519.

[93] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory", in *Advances in Neural Information Processing Systems*, 2018, pp. 4447–4458.

[94] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods", in *Advances in Neural Information Processing Systems*, 2018, pp. 5973–5983.

[95] A. F. Aji and K. Heafield, "Sparse Communication for Distributed Gradient Descent", in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 440–445.

[96] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training", in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=SkhQHMW0W.

[97] H. Sun, Y. Shao, J. Jiang, B. Cui, K. Lei, Y. Xu, and J. Wang, "Sparse gradient compression for distributed SGD", in *International Conference on Database Systems for Advanced Applications*, Springer, 2019, pp. 139–155.

[98] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization", in *Advances in Neural Information Processing Systems*, 2018, pp. 1299–1309.

[99] J. Jiang, F. Fu, T. Yang, and B. Cui, "SketchML: Accelerating distributed machine learning with data sketches", in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1269–1284.

[100] N. Ivkin, D. Rothchild, E. Ullah, V. braverman, I. Stoica, and R. Arora, "Communication-efficient distributed sgd with sketching", in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019.

[101] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright, "Atomo: Communication-efficient learning via atomic sparsification", in *Advances in Neural Information Processing Systems*, 2018, pp. 9850–9861.

[102] T. Vogels, S. P. Karimireddy, and M. Jaggi, "PowerSGD: Practical low-rank gradient compression for distributed optimization", in *Advances in Neural Information Processing Systems*, 2019, pp. 14 259–14 268.

[103] M. Cho, V. Muthusamy, B. Nemanich, and R. Puri, *GradZip: Gradient Compression using Alternating Matrix Factorization for Large-scale Deep Learning*, 2019.

[104] M. Yu, Z. Lin, K. Narra, S. Li, Y. Li, N. S. Kim, A. Schwing, M. Annavaram, and S. Avestimehr, "Gradiveq: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training", in *Advances in Neural Information Processing Systems*, 2018, pp. 5123–5133.

[105] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad, "Cloud computing pricing models: A survey", *International Journal of Grid and Distributed Computing*, vol. 6, no. 5, pp. 93–106, 2013.

[106] D. Kumar, G. Baranwal, Z. Raza, and D. P. Vidyarthi, "A survey on spot pricing in cloud computing", *Journal of Network and Systems Management*, vol. 26, no. 4, pp. 809–856, 2018.

[107] C. Wu, R. Buyya, and K. Ramamohanarao, "Cloud pricing models: Taxonomy, survey, and interdisciplinary challenges", *ACM Comput. Surv.*, vol. 52, no. 6, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3342103.

[108] M. Attaran and J. Woods, "Cloud computing technology: Improving small business performance using the internet", *Journal of Small Business & Entrepreneurship*, vol. 31, no. 6, pp. 495–519, 2019.

[109] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance", in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, IEEE, 2017, pp. 405–410.

[110] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.", Stanford InfoLab, Tech. Rep., 1999.

[111] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh, "Wtf: The who to follow service at twitter", in *Proceedings of the 22nd international conference on World Wide Web*, ACM, 2013, pp. 505–514.

[112] C. Ding and X. He, "K-means clustering via principal component analysis", in *Proceedings of the Twenty-First International Conference on Machine Learning*, ser. ICML ?04, Banff, Alberta, Canada, 2004, p. 29.

[113] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms", in *Proceedings of the 17th International Conference on Parallel Processing*, Springer, Bordeaux, France, 2011, pp. 90–109.

[114] R. A. van de Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm", Tech. Rep. 4, 1995, pp. 255–274.

[115] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes", *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5843–5855, 2014.

[116] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols", *Information Theory, IEEE Transactions on*, vol. 58, no. 11, pp. 6925–6934, Nov. 2012.

[117] H. Avron, K. L. Clarkson, and D. P. Woodruff, "Faster kernel ridge regression using sketching and preconditioning", *SIAM Journal on Matrix Analysis and Applications*, vol. 38, no. 4, pp. 1116–1138, 2017.

[118] A. Rahimi and B. Recht, "Random features for large-scale kernel machines", in *Advances in neural information processing systems*, 2008, pp. 1177–1184.

[119] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines", *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[120] P. Jain, P. Netrapalli, and S. Sanghavi, "Low-rank matrix completion using alternating minimization", in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, ACM, 2013, pp. 665–674.

[121] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems", *Computer*, no. 8, pp. 30–37, 2009.

[122] R. A. Sadek, "Svd based image processing applications: State of the art, contributions and research challenges", *arXiv preprint arXiv:1211.7102*, 2012.

[123] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, "Complex networks: Structure and dynamics", *Physics reports*, vol. 424, no. 4-5, pp. 175–308, 2006.

[124] P.-G. Martinsson, "A fast direct solver for a class of elliptic partial differential equations", *Journal of Scientific Computing*, vol. 38, no. 3, pp. 316–330, 2009.

[125] M. Powell, "Updating conjugate directions by the bfgs formula", *Mathematical Programming*, vol. 38, no. 1, p. 29, 1987.

[126] P. Sabino, "Monte carlo methods and path-generation techniques for pricing multi-asset path-dependent options", *arXiv preprint arXiv:0710.0850*, 2007.

[127] R. Eubank and S. Wang, "The equivalence between the cholesky decomposition and the kalman filter", *The American Statistician*, vol. 56, no. 1, pp. 39–43, 2002.

[128] S. L. Graham, M. Snir, and C. A. Patterson, *Getting up to speed: The future of supercomputing*. National Academies Press, 2005.

[129] L. I. Millett and S. H. Fuller, "Computing performance: Game over or next level?", *Computer*, vol. 44, pp. 31–38, Jan. 2011.

[130] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz, "Communication lower bounds and optimal algorithms for numerical linear algebra", *Acta Numerica*, vol. 23, pp. 1–155, 2014.

[131] M. Vijay and R. Mittal, "Algorithm-based fault tolerance: A review", *Microprocessors and Microsystems*, vol. 21, no. 3, pp. 151–161, 1997, Fault Tolerant Computing.

[132] J. Demmel, "Communication-avoiding algorithms for linear algebra and beyond", in *2013 IEEE 27th Int. Sym. on Parallel and Distributed Processing*, May 2013, pp. 585–585.

[133] A. Devarakonda, K. Fountoulakis, J. Demmel, and M. W. Mahoney, "Avoiding communication in primal and dual block coordinate descent methods", *arXiv:1612.04003*, 2016.

[134] S. P. Boyd, S. J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming", 2004.

[135] K. L. Clarkson and D. P. Woodruff, "Low rank approximation and regression in input sparsity time", in *Proc. of the Annual ACM Sym. on Theory of Computing*, Palo Alto, California, USA: ACM, 2013, pp. 81–90.

[136] S. Wang, "A practical guide to randomized matrix computations with matlab implementations", *arXiv:1505.07570*, 2015.

[137] X. Meng and M. W. Mahoney, "Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression", in *Proc. of the Forty-fifth Annual ACM Sym. on Theory of Computing*, Palo Alto, California, USA: ACM, 2013, pp. 91–100.

[138] N. Pham and R. Pagh, "Fast and scalable polynomial kernels via explicit feature maps", in *Proc. of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Chicago, Illinois, USA: ACM, 2013, pp. 239–247.

[139] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff, "Fast approximation of matrix coherence and statistical leverage", *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 3475–3506, Dec. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503308.2503352.

[140] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[141] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning", *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[142] N. S. Wadia, D. Duckworth, S. S. Schoenholz, E. Dyer, and J. Sohl-Dickstein, "Whitening and second order optimization both destroy information about the dataset, and can make generalization impossible", *arXiv e-prints*, arXiv:2008.07545, arXiv:2008.07545, Aug. 2020. arXiv: 2008.07545 [cs.LG].

[143] F. Roosta, Y. Liu, P. Xu, and M. W. Mahoney, "Newton-MR: Newton's method without smoothness or convexity", *arXiv preprint arXiv:1810.00303*, 2018.

[144] C.-H. Fang, S. B. Kylasa, F. Roosta-Khorasani, M. W. Mahoney, and A. Grama, "Distributed Second-order Convex Optimization", *ArXiv e-prints*, Jul. 2018. arXiv: 1807.07132.

[145] R. Anil, V. Gupta, T. Koren, K. Regan, and Y. Singer, "Second order optimization made practical", *arXiv preprint arXiv:2002.09018*, 2020.

[146] A. Gittens, A. Devarakonda, E. Racah, M. Ringenburg, L. Gerhardt, J. Kottalam, J. Liu, K. Maschhoff, S. Canon, J. Chhugani, *et al.*, "Matrix factorizations at scale: A comparison of scientific data analytics in spark and c+ mpi using three case studies", in *2016 IEEE International Conference on Big Data (Big Data)*, IEEE, 2016, pp. 204–213.

[147] V. Gupta, D. Carrano, Y. Yang, V. Shankar, T. Courtade, and K. Ramchandran, "Serverless straggler mitigation using local error-correcting codes", *IEEE International Conference on Distributed Computing and Systems (ICDCS), Singapore*, arXiv:2001.07490, arXiv:2001.07490, Jan. 2020. arXiv: 2001.07490 [cs.DC].

[148] R. Bollapragada, R. Byrd, and J. Nocedal, "Exact and Inexact Subsampled Newton Methods for Optimization", *arXiv e-prints*, arXiv:1609.08502, arXiv:1609.08502, Sep. 2016. arXiv: 1609.08502 [math.OC].

[149] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[150] A. S. Berahas, R. Bollapragada, and J. Nocedal, "An Investigation of Newton-Sketch and Subsampled Newton Methods", *arXiv e-prints*, arXiv:1705.06211, arXiv:1705.06211, May 2017. arXiv: 1705.06211 [math.OC].

[151] Y. Liu and F. Roosta, "Stability analysis of Newton-MR under hessian perturbations", *arXiv preprint arXiv:1909.06224*, 2019.

[152] J. R. Shewchuk *et al.*, *An introduction to the conjugate gradient method without the agonizing pain*, 1994.

[153] J. Levin, "Note on convergence of minres", *Multivariate behavioral research*, vol. 23, no. 3, pp. 413–417, 1988.

[154] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: An extension of mnist to handwritten letters", *arXiv preprint arXiv:1702.05373*, 2017.

[155] J. Nelson and H. L. Nguyen, "Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings", in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, Oct. 2013, pp. 117–126.

[156] T. Tao, *Topics in random matrix theory*. American Mathematical Soc., 2012, vol. 132.

[157] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, 1st ed. Springer Publishing Company, Incorporated, 2014.

[158] J. Acharya, C. De Sa, D. Foster, and K. Sridharan, "Distributed learning with sublinear communication", in *International Conference on Machine Learning*, 2019, pp. 40–50.

[159] Ghosh, Avishek, Maity, Rajkumar, Kadhe, Swanand, Mazumdar, Arya, and Ramachandran, Kannan, "Communication efficient and byzantine tolerant distributed learning", in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 2545–2550.

[160] Z. Dong, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision", *arXiv preprint arXiv:1905.03696*, pp. 293–302, 2019.

[161] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-BERT: Hessian based ultra low precision quantization of BERT", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 8815–8821.

[162] P. Mayekar and H. Tyagi, "Limits on Gradient Compression for Stochastic Optimization", *arXiv e-prints*, arXiv:2001.09032, arXiv:2001.09032, Jan. 2020. arXiv: `2001.09032 [cs.IT]`.

[163] B. Bartan and M. Pilanci, "Straggler resilient serverless computing based on polar codes", in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2019, pp. 276–283.

[164] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[165] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction", in *Advances in neural information processing systems*, 2013, pp. 315–323.

[166] S. Deorowicz and S. Grabowski, "Compression of DNA sequence reads in FASTQ format", *Bioinformatics*, vol. 27, no. 6, pp. 860–862, 2011.

[167] K. Vervier, P. Mahé, M. Tournoud, J. Veyrieras, and J. Vert, "Large-scale machine learning for metagenomics sequence classification", *Bioinformatics*, vol. 32, no. 7, pp. 1023–1032, 2016.

[168] A. Aghazadeh, A. Y. Lin, M. A. Sheikh, A. L. Chen, L. M. Atkins, C. L. Johnson, J. F. Petrosino, R. A. Drezek, and R. G. Baraniuk, "Universal microbial diagnostics using random DNA probes", *Sci. Adv.*, vol. 2, no. 9, e1600025, 2016.

[169] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch", in *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 29–42.

[170] N. Agarwal, B. Bullins, and E. Hazan, "Second-order stochastic optimization for machine learning in linear time", *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 4148–4187, Jan. 2017. [Online]. Available: `http://dl.acm.org/citation.cfm?id=3122009.3176860`.

[171] P. Xu, F. Roosta, and M. Mahoney, "Second-order optimization for non-convex machine learning: An empirical study", in *Proceedings of the 2020 SIAM International Conference on Data Mining*, SIAM, 2020, pp. 199–207.

[172] A. Mokhtari and A. Ribeiro, "Global convergence of online limited memory bfgs", *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 3151–3181, 2015.

[173] J. Nocedal, "Updating quasi-Newton matrices with limited storage", *Math. Comput.*, vol. 35, no. 151, pp. 773–782, 1980.

[174] D. Kane and J. Nelson, "Sparser Johnson-Lindenstrauss transforms", *J. ACM*, vol. 61, no. 1, p. 4, 2014.

[175] A. Maleki and D. L. Donoho, "Optimally tuned iterative reconstruction algorithms for compressed sensing", *IEEE J Selected Topics in Sig. Proces.*, vol. 4, no. 2, pp. 330–341, 2010.

[176] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "RCV1: A new benchmark collection for text categorization research", *J. Mach. Learn. Res.*, vol. 5, no. Apr, pp. 361–397, 2004.

[177] S. Webb, J. Caverlee, and C. Pu, "Introducing the Webb Spam Corpus: Using Email Spam to Identify Web Spam Automatically.", in *CEAS*, 2006.

[178] Y. Juan, Y. Zhuang, W. Chin, and C. Lin, "Field-aware factorization machines for CTR prediction", in *Proc.* 10^{th} *ACM Conf. Recommender Syst.*, ACM, 2016, pp. 43–50.

[179] J. A. Tropp, "An introduction to matrix concentration inequalities", *arxiv preprint arXiv: 1501.01571*, 2015.

[180] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp minima can generalize for deep nets", in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 1019–1028.

[181] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets", in *Advances in Neural Information Processing Systems*, 2018, pp. 6389–6399.

[182] S. Ma, R. Bassily, and M. Belkin, "The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning", *arXiv preprint arXiv:1712.06559*, pp. 3325–3334, 2017.

[183] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson, "Loss surfaces, mode connectivity, and fast ensembling of dnns", in *Advances in Neural Information Processing Systems*, 2018, pp. 8789–8798.

[184] S. Mandt, M. D. Hoffman, and D. M. Blei, "Stochastic gradient descent as approximate bayesian inference", *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 4873–4907, 2017.

[185] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout", *arXiv preprint arXiv:1708.04552*, 2017.

[186] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Dawnbench: An end-to-end deep learning benchmark and competition",

[187] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in tensorflow", *arXiv preprint arXiv:1802.05799*, 2018.

[188] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et al.*, "Large scale distributed deep networks", in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[189] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server", in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 583–598.

[190] C. H. Martin and M. W. Mahoney, "Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning", Tech. Rep. Preprint: arXiv:1810.01075, 2018.

[191] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis", *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[192] Z. Yao, A. Gholami, Q. Lei, K. Keutzer, and M. W. Mahoney, "Hessian-based analysis of large batch training and robustness to adversaries", in *Advances in Neural Information Processing Systems*, 2018, pp. 4949–4959.

[193] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer", *arXiv preprint arXiv:1701.06538*, 2017.

[194] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[195] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, *et al.*, "Deep learning recommendation model for personalization and recommendation systems", *arXiv preprint arXiv:1906.00091*, 2019.

[196] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling", in *2011 38th Annual international symposium on computer architecture (ISCA)*, IEEE, 2011, pp. 365–376.

[197] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module GPUs for continued performance scalability", *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 320–332, 2017.

[198] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "PipeDream: generalized pipeline parallelism for DNN training", in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[199] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and z. Chen, "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism", in *Advances in Neural Information Processing Systems 32*, 2019, pp. 103–112.

[200] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform", *arXiv preprint arXiv:1809.02839*, 2018.

[201] J. Geng, D. Li, and S. Wang, "ElasticPipe: An Efficient and Dynamic Model-Parallel Solution to DNN Training", in *Proceedings of the 10th Workshop on Scientific Cloud Computing*, ser. ScienceCloud '19, Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 5–9.

[202] L. Guan, W. Yin, D. Li, and X. Lu, "XPipe: Efficient Pipeline Model Parallelism for Multi-GPU DNN Training", *arXiv preprint arXiv:1911.04610*, 2019.

[203] J. K. Kim, Q. Ho, S. Lee, X. Zheng, W. Dai, G. A. Gibson, and E. P. Xing, "STRADS: A Distributed Framework for Scheduled Model Parallel Machine Learning", ser. EuroSys '16, London, United Kingdom: Association for Computing Machinery, 2016.

[204] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "A hierarchical model for device placement", in *International Conference on Learning Representations*, 2018.

[205] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks", 2019.

[206] J. Geng, D. Li, and S. Wang, "Horizontal or vertical? a hybrid approach to large-scale distributed machine learning", in *Proceedings of the 10th Workshop on Scientific Cloud Computing*, 2019, pp. 1–4.

[207] J. H. Park, G. Yun, C. M. Yi, N. T. Nguyen, S. Lee, J. Choi, S. H. Noh, and Y.-r. Choi, "HetPipe: Enabling Large DNN Training on (Whimpy) Heterogeneous GPU Clusters through Integration of Pipelined Model Parallelism and Data Parallelism", *arXiv preprint arXiv:2005.14038*, 2020.

[208] Z. Jia, S. Lin, C. R. Qi, and A. Aiken, "Exploring hidden dimensions in parallelizing convolutional neural networks", *arXiv preprint arXiv:1802.04924*, 2018.

[209] A. Gholami, A. Azad, P. Jin, K. Keutzer, and A. Buluc, "Integrated model, batch, and domain parallelism in training neural networks", in *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, 2018, pp. 77–86.

[210] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD", *preprint arXiv:1604.00981*, 2016.

[211] S. Lee, J. K. Kim, X. Zheng, Q. Ho, G. A. Gibson, and E. P. Xing, "On model parallelization and scheduling strategies for distributed machine learning", in *Advances in neural inf. processing systems*, 2014, pp. 2834–2842.

[212] H. Rong, Y. Wang, F. Zhou, J. Zhai, H. Wu, R. Lan, F. Li, H. Zhang, Y. Yang, Z. Guo, *et al.*, "Distributed Equivalent Substitution Training for Large-Scale Recommender Systems", in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 911–920.

[213] B. Yang, J. Zhang, J. Li, C. Ré, C. R. Aberger, and C. De Sa, "PipeMare: Asynchronous Pipeline Parallel DNN Training", *arXiv preprint arXiv:1910.05124*, 2019.

[214] A. Makhzani and B. Frey, "K-sparse autoencoders", *arXiv preprint arXiv:1312.5663*, 2013.

[215] A. Makhzani and B. J. Frey, "Winner-take-all autoencoders", in *Advances in neural information processing systems*, 2015, pp. 2791–2799.

[216] B. Chen, T. Medini, J. Farwell, S. Gobriel, C. Tai, and A. Shrivastava, "Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems", *preprint arXiv:1903.03129*, 2019.

[217] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[218] J. Ba and B. Frey, "Adaptive dropout for training deep neural networks", in *Advances in neural information processing systems*, 2013, pp. 3084–3092.

[219] M. W. Mahoney, "Approximate Computation and Implicit Regularization for Very Large-scale Data Analysis", in *Proceedings of the 31st ACM Symposium on Principles of Database Systems*, 2012, pp. 143–154.

[220] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[221] J. Shun, F. Roosta-Khorasani, K. Fountoulakis, and M. W. Mahoney, "Parallel local graph clustering", *Proc. of the VLDB Endowment*, vol. 9, no. 12, pp. 1041–1052, 2016.

[222] K. Fountoulakis, D. F. Gleich, and M. W. Mahoney, "An optimization approach to locally-biased graph algorithms", *Proceedings of the IEEE*, vol. 105, no. 2, pp. 256–272, 2017.

[223] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press, 2016, vol. 1.

[224] M. Cettolo, C. Girardi, and M. Federico, "Wit3: Web inventory of transcribed and translated talks", in *Conference of european association for machine translation*, 2012, pp. 261–268.

[225] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A Fast, Extensible Toolkit for Sequence Modeling", in *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[226] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need", in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[227] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent", in *Advances in neural information processing systems*, 2011, pp. 693–701.

[228] P. Drineas and M. W. Mahoney, "RandNLA: Randomized numerical linear algebra", *Communications of the ACM*, vol. 59, pp. 80–90, 2016.

[229] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider", *arXiv e-prints*, arXiv:2003.03423, arXiv:2003.03423, Mar. 2020. arXiv: `2003.03423 [cs.DC]`.

[230] P. Patel, A. H. Ranabahu, and A. P. Sheth, "Service level agreement in cloud computing", 2009.

[231] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks", *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 50–61, 2011.

[232] W. Wang, P. Zhang, T. Lan, and V. Aggarwal, "Datacenter net profit optimization with deadline dependent pricing", in *2012 46th Annual Conference on Information Sciences and Systems (CISS)*, IEEE, 2012, pp. 1–6.

[233] C. Joe-Wong and S. Sen, "Mathematical frameworks for pricing in the cloud: Revenue, fairness, and resource allocations", *arXiv preprint arXiv:1212.0022*, 2012.

[234] R. B. Halima, S. Kallel, W. Gaaloul, and M. Jmaiel, "Optimal cost for time-aware cloud resource allocation in business process", in *2017 IEEE International Conference on Services Computing (SCC)*, IEEE, 2017, pp. 314–321.

[235] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing amazon ec2 spot instance pricing", *ACM Transactions on Economics and Computation (TEAC)*, vol. 1, no. 3, pp. 1–20, 2013.

[236] F. Kelly, "Charging and rate control for elastic traffic", *European transactions on Telecommunications*, vol. 8, no. 1, pp. 33–37, 1997.

[237] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability", *Journal of the Operational Research society*, vol. 49, no. 3, pp. 237–252, 1998.

[238] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control", *IEEE/ACM Transactions on networking*, vol. 8, no. 5, pp. 556–567, 2000.

[239] S. H. Low and D. E. Lapsley, "Optimization flow control. i. basic algorithm and convergence", *IEEE/ACM Transactions on networking*, vol. 7, no. 6, pp. 861–874, 1999.

[240] S. Kunniyur and R. Srikant, "End-to-end congestion control schemes: Utility functions, random losses and ecn marks", *IEEE/ACM Transactions on networking*, vol. 11, no. 5, pp. 689–702, 2003.

[241] R. McNaughton, "Scheduling with deadlines and loss functions", *Management Science*, vol. 6, no. 1, pp. 1–12, 1959.

[242] T. Achterberg, "Scip: Solving constraint integer programs", *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.

[243] M. Wang, C. W. Tan, W. Xu, and A. Tang, "Cost of not splitting in routing: Characterization and estimation", *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1849–1859, 2011.

[244] V. L. Smith, "Experimental auction markets and the walrasian hypothesis", *Journal of Political Economy*, vol. 73, no. 4, pp. 387–393, 1965.

[245] A. Mas-Colell, M. D. Whinston, *et al.*, *Microeconomic theory*, vol. 1.

[246] S.-P. Han, "A successive projection method", *Mathematical Programming*, vol. 40, no. 1, pp. 1–14, 1988.

[247] S. R. Phade and V. S. Borkar, "A distributed boyle–dykstra–han scheme", *SIAM Journal on Optimization*, vol. 27, no. 3, pp. 1880–1897, 2017.

[248] S. Phade and V. Anantharam, "Optimal resource allocation over networks via lottery-based mechanisms", in *International Conference on Game Theory for Networks*, Springer, 2019, pp. 51–70.

[249] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time operating systems.", in *Rtss*, vol. 85, 1985, pp. 112–122.

[250] D. Vengerov, "Adaptive utility-based scheduling in resource-constrained systems", in *Australasian Joint Conference on Artificial Intelligence*, Springer, 2005, pp. 477–488.

[251] H. Wu, B. Ravindran, E. D. Jensen, and U. Balli, "Utility accrual scheduling under arbitrary time/utility functions and multi-unit resource constraints", *Proc. of the 10th Real-Time and Embedded Computing Systems and Applications*, p. 1, 2004.

[252] R. K. Clark, "Scheduling dependent real-time activities", CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, Tech. Rep., 1990.

[253] D. A. Menasce and M. N. Bennani, "Autonomic virtualized environments", in *International conference on autonomic and autonomous systems (ICAS'06)*, IEEE, 2006, pp. 28–28.

[254] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility functions in autonomic systems", in *International Conference on Autonomic Computing, 2004. Proceedings.*, IEEE, 2004, pp. 70–77.

[255] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers", *ACM SIGOPS operating systems review*, vol. 35, no. 5, pp. 103–116, 2001.

[256] D. Minarolli and B. Freisleben, "Utility-based resource allocation for virtual machines in cloud computing", in *2011 IEEE symposium on computers and communications (ISCC)*, IEEE, 2011, pp. 410–417.

[257] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud", *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.

[258] C. Weinhardt, A. Anandasivam, B. Blau, N. Borissov, T. Meinl, W. Michalk, and J. Stößer, "Cloud computing–a classification, business models, and research directions", *Business & Information Systems Engineering*, vol. 1, no. 5, pp. 391–399, 2009.

[259] H. Li, J. Liu, and G. Tang, "A pricing algorithm for cloud computing resources", in *2011 international conference on network computing and information security*, IEEE, vol. 1, 2011, pp. 69–73.

[260] J. Rohitratana and J. Altmann, "Impact of pricing schemes on a market for software-as-a-service and perpetual software", *Future Generation Computer Systems*, vol. 28, no. 8, pp. 1328–1339, 2012.

[261] M. Mihailescu and Y. M. Teo, "Dynamic resource pricing on federated clouds", in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, IEEE, 2010, pp. 513–517.

[262] P. Milgrom and I. Segal, "Envelope theorems for arbitrary choice sets", *Econometrica*, vol. 70, no. 2, pp. 583–601, 2002.

[263] J. Du and J. Y.-T. Leung, "Minimizing total tardiness on one machine is np-hard", *Mathematics of operations research*, vol. 15, no. 3, pp. 483–495, 1990.

[264] E. L. Lawler, "A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness", in *Annals of discrete Mathematics*, vol. 1, Elsevier, 1977, pp. 331–342.

[265] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems", *Studies in integer programming*, vol. 1, pp. 343–362, 1977.

[266] V. Gupta, S. Phade, T. Courtade, and K. Ramchandran, "Utility-based resource allocation and pricing for serverless computing", *arXiv preprint arXiv:2008.07793*, 2020.

[267] A. Tversky and D. Kahneman, "Advances in prospect theory: Cumulative representation of uncertainty", *Journal of Risk and uncertainty*, vol. 5, no. 4, pp. 297–323, 1992.

[268] P. P. Wakker, *Prospect theory: For risk and ambiguity*. Cambridge University Press, 2010.

[269] R. Gonzalez and G. Wu, "On the shape of the probability weighting function", *Cognitive psychology*, vol. 38, no. 1, pp. 129–166, 1999.

[270] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.

[271] C. Starmer, "Developments in non-expected utility theory: The hunt for a descriptive theory of choice under risk", *Journal of economic literature*, vol. 38, no. 2, pp. 332–382, 2000.

[272] A. Chateauneuf and P. Wakker, "An axiomatization of cumulative prospect theory for decision under risk", *Journal of Risk and Uncertainty*, vol. 18, no. 2, pp. 137–145, 1999.

[273] D. Kahneman and A. Tversky, "Prospect theory: An analysis of decision under risk", in *Handbook of the fundamentals of financial decision making: Part I*, World Scientific, 2013, pp. 99–127.