

Hold 'em or Fold 'em? Aggregation Queries under Performance Variations

*Gautam Kumar
Ganesh Ananthanarayanan
Sylvia Ratnasamy
Ion Stoica*



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-267

<http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-267.html>

December 21, 2015

Copyright © 2015, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Hold 'em or Fold 'em? Aggregation Queries under Performance Variations

Gautam Kumar

Ganesh Ananthanarayanan

Sylvia Ratnasamy

Ion Stoica

Abstract

Systems are increasingly required to provide responses to queries, even if not exact, within stringent time deadlines. These systems parallelize computations over many *processes* and *aggregate* them hierarchically to get the final response (e.g., search engines and data analytics). Due to large performance variations in clusters, some processes are slower. Therefore, aggregators are faced with the question of how long to wait for outputs from processes before combining and sending them upstream. Longer waits increase the *response quality* as it would include outputs from more processes. However, it also increases the risk of the aggregator failing to provide its result by the deadline. This leads to all its results being ignored, degrading response quality. Our algorithm, Cedar, proposes a solution to this quandary of deciding wait durations at aggregators. It uses an online algorithm to learn distributions of durations at each level in the hierarchy and collectively optimizes the wait duration. Cedar's solution is theoretically sound, fully distributed, and generically applicable across systems that use aggregation trees since it is agnostic to the causes of performance variations. Evaluation using production latency distributions from Google, Microsoft and Facebook using deployment and simulation shows that Cedar improves average response quality by over 100%.

1 Introduction

Systems using *aggregation trees* in their computation are increasingly pervasive (e.g., web search engines and approximate querying frameworks [1]). These computations have many parallel *processes* with *aggregators* arranged hierarchically to combine their outputs. Figure 1 shows a simple abstract illustration. Modern systems that use aggregation trees run on large clusters and are required to provide responses to queries, *even if inexact*, within stringent time deadlines [2, 3, 1, 4].

Endemic to large clusters are broad performance variations, resulting in some processes being much slower than others. These variations can arise due to network congestion [5, 6, 2] as well as systemic contentions [7, 8, 9, 10]. For instance, production traces show that RTT values in Bing's search cluster can vary by a factor of nearly $50\times$ [6].

Due to slow processes, every aggregator faces the decision of how long to *wait* for outputs from processes be-

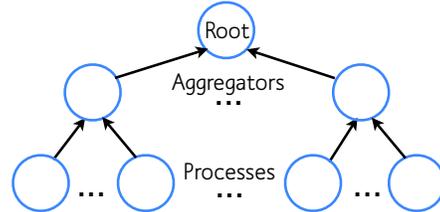


Figure 1: **Aggregation Trees.** On arrival of a *query*, computations are spawned across multiple parallel *processes* whose outputs are combined using *aggregators* to produce the final response.

fore aggregating and sending the results upstream. The wait duration has direct implication on the *quality* of the response. We define *response quality* as the *fraction of process outputs that are included in the response*; similar definitions of quality (or application throughput) have been used in many recent proposals [2, 11]. The longer the aggregator waits, the higher the quality of the overall response. However, longer wait durations also increase the risk of the aggregator failing to provide its results to the root by the deadline. If it misses its deadline, all its results are ignored by the root, thus lowering quality of the final response.¹ In this paper, we ask the question:

How long should every aggregator wait to maximize overall response quality within the deadline?

The main observation behind our solution is that the durations of the queries follow a certain distribution type, and that it is possible to quite accurately compute the parameters of this distribution by observing a few process outputs. In particular, we use the durations of the processes that finish first to predict the durations' distribution, and use this distribution to optimize the response quality by appropriately setting the wait-time for the aggregator.

However any approach to learn the distribution parameters must overcome the following challenges. First, learning must be done in an online fashion on a per-query basis given that different queries can vary substantially owing to the different amount of work they might need to perform. Second, given that we need to learn the dis-

¹While aggregators can periodically update the root with their results, systems avoid such a design because, (i) it increases network load by a factor as much as the number of updates, and (ii) it complicates the root and aggregator executions along with their failure semantics. Production systems, to the best of our best knowledge, avoid such periodic updates.

tribution parameters online, the approach must be based on only the earliest completed processes. Naturally, this introduces a bias as the learning will not see the tail of the distribution, *i.e.*, the outputs of the processes that take longest to complete. Third, since we also target systems with short deadlines ($\sim 100 - 200\text{ms}$), the learning must be distributed, *i.e.*, not require aggregators to combine their samples, to avoid communication overheads.

To address the above challenges, we propose Cedar, an online algorithm to pick the wait duration for each aggregator. Cedar learns the distribution of process durations during a query’s execution using statistically grounded techniques. It avoids the measurement bias due to observing only the earliest available process outputs by using the properties of *order statistics* [12]. Once it learns the distribution of process durations, Cedar picks a wait duration based on the query’s end-to-end deadline, as well as the time taken by aggregators themselves to combine and send the results to the root.

Since Cedar learns the distribution parameters with high accuracy even using a small number of samples, each aggregator can estimate the parameters standalone without pooling their samples. Thus, Cedar can be implemented in a *fully distributed* manner.

Cedar’s solution has the following key advantages.

- It considers the problem of deadline-aware scheduling *end-to-end*, *i.e.*, aiming to improve application performance instead of just individual processes.
- It makes no assumptions about the source of performance variations among processes nor does it attempt to mitigate such variations. This generality differentiates our solution from many prior efforts that assume specific causes like network congestion [2, 7, 11]. Such generality is critical for any solution to work well in practice because there is no single cause for performance variations and accurate modeling of these complex systems has proven challenging so far [10, 13, 9, 14]. In this way, Cedar’s performance benefits are agnostic to the cause of these variations, whether they occur because of CPU, memory, network or disk contention.
- Unlike many prior solutions that require changes at the network layer [5, 2, 11], Cedar can be implemented entirely at the endhosts. This leads to a simpler and easily deployable solution. Furthermore, Cedar is robust across different workloads.

To the best of our knowledge, prior work on dealing with performance variations has not explored optimizing the wait duration at aggregators. Optimizing along this simple design dimension leads to remarkably good results in our experiments. We evaluate Cedar using a prototype implementation over the Spark framework [15]

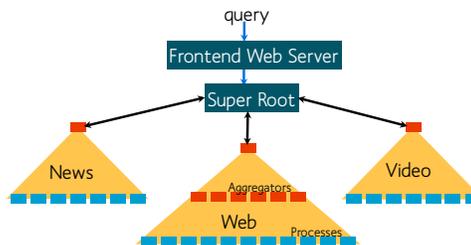


Figure 2: A typical web search computation that aggregates results across many functional silos [10]. The small rectangles at the leaves denote processes of the computation.

deployed on 80 machines on EC2, as well as through extensive simulations. Cedar improves the quality of results by up to 100% in simulations and deployment replaying production traces from Facebook and Bing’s analytics clusters as well as Google and Bing’s search clusters. The near-optimal performance is due to accurate learning of distribution parameters using order statistics leading to as little as 5% estimation error as well as a theoretically sound algorithm to select the correct wait duration given the distributions.

2 Aggregation Queries

Aggregation queries are widely prevalent in modern systems. We describe two such systems—web services and data analytics frameworks—that strive to provide results of the best quality (but not necessarily exact) within a deadline. We then quantify performance variations in production clusters that run these systems.

2.1 Production Systems

Search Queries: Web search engines store indexes of crawled web content on large distributed clusters. The indexes are often divided into functional silos. To respond to a search query, lookups are performed on different machines within every relevant silo, effectively resulting in a computation of many parallel *processes* whose outputs are aggregated hierarchically (as shown in Figure 2). Every aggregator ranks results from nodes downstream and sends the top few of them upstream. The eventual response is based on results that arrive at the root by the deadline. The higher the number of processes whose outputs are included in the response, the better its quality and relevance [2], which in turn has significant competitive and financial implications [16]. Similar hierarchical computations are also invoked in the creation of a user’s “wall” page in Facebook [5].

Typically, process durations are primarily influenced by contentions for multiple local resources as they read and compute on indexes from the underlying storage. Aggregator durations, on the other hand, are influenced more by networking and scheduling aspects.

Approximate Analytics: Interactive data analytics

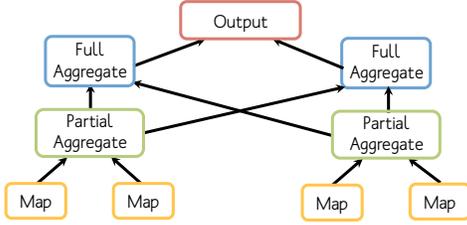


Figure 3: A query to an interactive data analytics system compiled into a DAG of hierarchical parallel processes.

frameworks (*e.g.*, Dremel [17], BlinkDB [1]) are projected to be crucial in exploiting the ever growing data. These frameworks allow users to specify deadlines in the query syntax, and they strive to provide the best quality response within that deadline [3, 2]. Queries are compiled to a DAG of phases (or hierarchies) where each phase consists of multiple parallel processes. Figure 3 shows a DAG of a sample query. While the communication pattern between hierarchies can be either many-to-one or all-to-all, every aggregator aggregates results from nodes downstream and sends them upstream. The quality of responses is, again, dictated by the number of processes whose outputs are included in the response.

2.2 Performance Variations

The nature of large clusters is that processes exhibit significant and unpredictable variations in their completion. These variations arise due to network congestion or contention for resources on individual machines. We present variations (in increasing order of magnitude) from three production deployments—RTT variations in Microsoft Bing’s search cluster [6], process durations in Google’s search cluster [10], and task completion times in Facebook’s and Bing’s production analytics cluster [18]. These clusters already have a variety of mitigation strategies to prevent processes from straggling [5, 6, 7, 8, 9].

The objective of describing these variations is two fold. First, is to show the prevalence and magnitude of performance variations, and also that they occur due to multiple reasons. Second, is to present building blocks for constructing a workload for aggregation queries in the absence of access to an end-to-end trace from systems as described in §2.1.

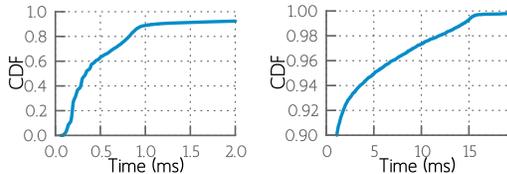


Figure 4: Distribution of RTTs in Bing’s search cluster. The median value is $330\mu\text{s}$ while the 90th and 99th percentile values are 1.1ms and 14ms, respectively.

Figure 4 plots the distribution of RTT values in Bing’s

search cluster. The RTT values show a long-tailed distribution (median, 90th percentile and 99th percentile values of $330\mu\text{s}$, 1.1ms and 14ms) caused due to sporadic network congestion. Variations among processes in Google’s search cluster are primarily borne out of scheduling delays and network congestion on highly utilized machines/links [10]. While the distribution is relatively narrow, the magnitude of the variation is significantly higher. The median value is 19ms while the 99th percentile is over 65ms.

The task durations in Facebook’s and Bing’s analytics clusters vary considerably more (factor of $1600\times$) and are caused by a combination of systemic contention for local resources (memory, CPU and disk IO). Note that these clusters already have speculation strategies [7, 8] for stragglers. When the earlier of the original or speculative copies finish, the unfinished task is killed; we exclude durations of such killed tasks. Further, task durations have recently fallen by a factor of two to three orders of magnitude with the advent of in-memory cluster frameworks (*e.g.*, Spark [15], Dremel [17]). At small task durations, effectiveness of existing straggler mitigation strategies are diminished owing to their reactive nature of observing a straggler before scheduling speculative copies [9].

The upshot from these production traces is that performance variations are large and occur due to many reasons. Ideally, algorithms to decide wait durations at aggregators should take a holistic end-to-end view of the variations and automatically adapt to any changes in the distributions, without being tied to the underlying specifics. Before proceeding to our solution, Cedar, in §4, we quantify the criticality of picking the right wait-duration in §3 using an idealized scheme.

3 The Case for Optimizing Wait Duration

We illustrate the value of picking the optimal wait duration at aggregators by comparing the difference in response quality between an ideal scheme and intuitive straw-man solutions. We focus on the traces from Facebook’s Analytics (MapReduce) cluster in this section, though our evaluations (§5) are based on a number of production and synthetic workloads.

We assume a two level (or stage) hierarchy as shown in Figure 5, where X_1 and X_2 denote the distribution of times taken by nodes in the first and the second levels of the hierarchy, respectively, with k_1 and k_2 being the “fan out” at these levels.² There is an end-to-end deadline, D , imposed on the query which is common knowledge to workers and aggregators alike. However, while the aggregators know the top-level deadline, they can’t exactly

²Two or three levels are common in the systems we focus on. However, we will show later that our model works with any number of levels.

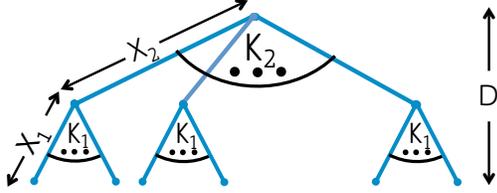


Figure 5: A simple two level hierarchy. Let X_1 and X_2 be distribution of times taken in the first and the second levels of the hierarchy, respectively.

determine how long it will take to ship the aggregated result upstream. As mentioned in §1, one of the strengths of our model is that since X_1 and X_2 represent the duration of the entire stage *subsuming all types of variations*, the performance improvement doesn't depend on their exact cause, be it network, CPU, memory or disk.

3.1 Setting Wait Durations

Recall that wait duration at aggregators directly impacts *overall response quality*; we measure quality by the fraction of processes whose outputs are included in the final result.³ If the wait duration is too short, outputs that would have arrived before the deadline are missed thereby degrading overall response quality. If the wait duration is too large, the aggregator misses its deadline which leads to outputs of all its processes (including the completed ones) being ignored upstream, again degrading overall response quality. Therefore, our problem is to *calculate the right wait duration at aggregators that maximizes overall response quality given a deadline of D .*

Proportional-split: A natural straw-man solution to pick the wait duration is to continuously learn statistics about the underlying distributions X_1 and X_2 from completed queries, and split the deadline proportionally between the different levels based on the learned parameters. In fact, such a technique of estimating parameters from recent query behavior is in deployment in Google's clusters [4]. For a two-level tree (Figure 5), the wait duration is set as $D \times \left(\frac{\mu(X_1)}{\mu(X_1) + \mu(X_2)} \right)$, where $\mu(X_1)$ is the mean of the stage duration distribution X_1 . We refer to such a scheme as "Proportional-split".⁴

Ideal Solution: We compare the Proportional-split baseline with an idealized scheme that has *a priori* information about the distribution of process as well as aggregator durations of every query. It uses that information to pick the wait duration that maximizes overall response quality. We measure the *percentage improvement in response quality of the idealized scheme over the straw-*

³Our model is easily extensible to weighted process outputs (Appendix A).

⁴Other statistics like median and (mean + stdev) exhibit similar results. Further, we also considered other baselines like equally dividing the deadline between the stages or subtracting the mean of X_2 from the deadline, but they fare much worse.

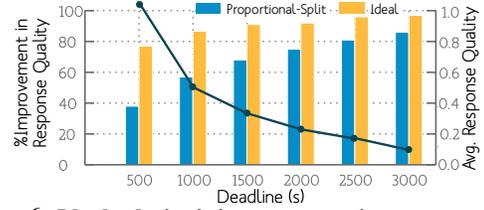


Figure 6: Ideal solution's improvement in response quality over straw-man solutions for varying deadlines. Distribution X_1 is from the map tasks and X_2 is from the reduce tasks in the Facebook cluster. The fanout is kept constant at 50 (both k_1 and k_2) giving a total of 2500 processes.

man solutions. The ideal scheme serves as a ceiling to estimate the maximum achievable improvement; the higher the improvement, the more the potential for Cedar.

3.2 Potential Gains

Recall from §2.2 that the clusters from which we obtain our traces already have a variety of straggler mitigation strategies deployed. Despite that, we see substantial scope for improvement in quality of the overall response between the Proportional-split and the Ideal scheme. In Figure 6, the deadline for the query is varied from 500s to a really high value of 3000s while the fanout factors k_1 and k_2 are kept constant at 50 (based on [6]). Picking the right wait duration can improve average response quality by over 100% compared to Proportional-split, *i.e.*, the response includes outputs of 100% more processes. Also, while it is *ideally* possible to achieve high response quality (say 90%) at deadline values of > 1000 s, the baseline fails to achieve such response quality even at an extremely large deadline of 3000s. Note that Proportional-split, even though it knows the distributions, ignores query-specific variations since it uses a single distribution over the recent set of queries. Overall, potential for such high gains, despite the presence of straggler mitigation strategies, shows the criticality of setting the right wait duration.

Summary: Our analysis shows that setting the right wait duration for aggregators (*i*) has substantial potential to improve response quality (by over 100%), and (*ii*) is non-trivial and simple straw-man solutions fall significantly short.

4 Cedar: Algorithm Description

The aggregator estimates the optimal wait time by learning the distribution parameters of process durations during a query's execution. The two main steps in doing so are, (*i*) learning the distribution based on the completion times of only the early processes in the query, and (*ii*) collectively optimizing for the wait duration taking into account the time taken by aggregators themselves to combine and send the results to the root. We describe these steps after presenting an overview.

```

(⟨ProcessOutput⟩ outputSet ←  $\phi$  ▷ Response Initialization
numOutputs ← 0

```

```

procedure PARALLELHIERARCHICALCOMP( $D$ )
  SetTimer( $D$ , TIMEREXPIRE)
  ListenResponse(PROCESSHANDLER)

```

```

procedure PROCESSHANDLER(ProcessOutput  $t_o$ )
  ▷ On arrival of a process’s output
  outputSet +=  $t_o$  ▷  $t_o$  added to outputSet
  numOutputs += 1
  if numOutputs ==  $k_1$  then
    SetTimer(0, TIMEREXPIRE); return

```

```

Distribution  $X_1$  ← FITDISTRIBUTION(outputSet)
double wait ← CALCULATEWAIT( $D$ ,  $k_1$ ,  $X_1$ )
  ▷ Also uses Distributions  $X_i$ ’s and  $k_i$ ’s for higher
  levels (global)

```

```

double remWait ← wait − elapsedTime
  ▷ Subtract time elapsed so far.
  SetTimer(remWait, TIMEREXPIRE)
  ListenResponse(PROCESSHANDLER)

```

```

procedure TIMEREXPIRE
  return outputSet

```

Pseudocode 1: Cedar’s algorithm for executing aggregation queries with deadline of D . The algorithm describes the functioning of an aggregator at the lowest layer with k_1 processes whose durations are modeled as X_1 , aggregators up the hierarchy work similarly. FITDISTRIBUTION is described in §4.2 to estimate X_1 and CALCULATEWAIT is described in §4.3.

4.1 Overview

Pseudocode 1 outlines Cedar’s end-to-end functioning. The aggregator begins by setting a timer for the deadline, D (PARALLELHIERARCHICALCOMP). On arrival of every process’s output (PROCESSHANDLER), Cedar improves its estimation of the distribution (FITDISTRIBUTION) and updates its wait duration (CALCULATEWAIT). The aggregator returns with the available outputs when no process finishes in its current wait duration (TIMEREXPIRE).

Typically, higher levels in the hierarchy, *i.e.*, aggregators, have little variation in the distribution of their durations across queries (X_2 for the two-stage tree demonstrated in Figure 5). This is because aggregation operations are mostly similar across different queries (for example, sum and mean, which have similar complexities). These trends are observed in our analysis of traces from Google and Bing. These two traces are primarily from higher level aggregator operations and exhibit little variation across queries. Thus, Cedar learns the above stage distributions offline based on completed queries.

FITDISTRIBUTION concerns itself with learning the

Table 1: Table of Notation

D	\triangleq	Deadline at the top-level aggregator
n	\triangleq	Number of stages in the aggregation tree
X_i	\triangleq	Stage duration distribution for the i^{th} stage (1 being bottom-most)
k_i	\triangleq	Fan-out at the i^{th} stage
X, k	\triangleq	Stage duration distribution and fan-out when a single stage is considered
$X_{(i)}$	\triangleq	i^{th} -order statistic of X
q_n	\triangleq	Maximum achievable quality for an n -level tree (under given D , X_i ’s and k_i ’s).

distributions of process durations, X_1 . As evidenced in the Facebook distributions, process durations exhibit significant variation across queries. Processes execute custom code that involve a wide variety of compute and IO operations (across disk, network) leaving them susceptible to many resource contentions. As an illustration, the computation involved for a search query like “Britney Spears” may take considerably lesser time compared to a more sophisticated query like “Britney Spears Grammy Toxic” because the latter involves a combination of index lookups. Therefore, it becomes imperative to determine the distribution of process durations per query.

CALCULATEWAIT, then, uses both X_1 and higher level distributions, X_2, \dots, X_n , to calculate the optimal wait duration for every aggregator.

We explain the learning of distribution of process durations (FITDISTRIBUTION) in §4.2. §4.3 explains the calculation of the optimal wait duration for aggregators (CALCULATEWAIT). Table 1 lists the relevant notations.

4.2 Learning the Distribution

Cedar estimates the parameters of the distribution of process durations *online* during the query’s execution. The estimation involves two aspects—distribution type (*e.g.*, log-normal, exponential), and relevant distribution parameters (*e.g.*, mean and standard deviation).

4.2.1 Distribution Type

Inspection of process durations from our traces show that the distribution type across different queries remains unchanged, even though the parameters of the distribution vary. Therefore, estimating the distribution type is an offline process that is repeated periodically across many completed queries. We periodically fit percentile values using *rriskDistributions* [19] package to find the best fit of distribution type.

In our traces, log-normal distribution gave the best fit for each of the traces. The fit for the Facebook traces resulted in less than 1% error in mean and median; even at high percentiles the error was low. Google’s percentile values for process durations fit with < 5% error even at the 99th percentile. Log-normal distribution gave the best

fit for the Bing traces as well with 1% error in median and 2% error in mean. More details about the goodness of our fit can be found in Appendix B.⁵

Regardless of the distribution type, its parameters exhibit considerable variation on a query to query basis. We estimate them online, during the query’s execution.

4.2.2 Estimating Parameters of Distribution

We consider a single level in which an aggregator observes process duration distribution given by X with a fan-out of k (omitting the subscript that denotes the level since we are concerned with a single stage in this section). The objective is to estimate the parameters of the distribution X based on the durations of only the completed processes thus far. Denote the number of processes that have completed at this point to be r , where $r < k$. A naive attempt at estimating the parameters of the distribution would be to simply calculate the mean and standard deviation using the r samples. Such an empirical calculation would, however, be sub-optimal because the r samples are *biased*, i.e., these are not uniformly sampled r values from the distribution but rather the smallest r values out of k samples from the distribution. The key challenge in learning the parameters of the distribution is, therefore, eradicating this bias. As we show in §5, this bias can affect the accuracy of learnt parameters considerably.

We alleviate the sampling bias using *order statistics* [12]. Given a distribution, X in our case, and k random samples drawn from the distribution, the r^{th} order statistic denotes the distribution of the random variable corresponding to the r^{th} -minimum of the k samples. The key insight that Cedar uses is that the time taken for the r^{th} process output received is not a random sample obtained from the distribution X , but instead, is a random sample obtained from the r^{th} order-statistic of the k samples drawn from X .⁶ In this way, Cedar models *each* process duration as per a different distribution which are given by the *order statistics* for the given distribution.

Formally, denote the random variables corresponding to the first r order statistics (or process durations in our case) by $X_{(1)}, X_{(2)}, \dots, X_{(r)}$ (the subscript denotes the order they arrive in),⁷ and let $x_{(1)}, x_{(2)}, \dots, x_{(r)}$ be the observed values of process durations for the received outputs. The maximum likelihood estimate, θ , of the distribution pa-

⁵One concern is that log-normal fit does seem to falter near the extreme tail (say upwards of 99.5 percentile); the tail being generally better modeled by distributions like Pareto [20]. Such high percentiles, however, would consist of processes whose outputs will *not* be aggregated irrespective of any optimization of wait-duration given the heavy-tail behavior of such systems. Thus Cedar’s performance doesn’t suffer due to this and remains near-optimal (§5).

⁶Order statistics are dependent on the sample size (or k in our case).

⁷ $X_{(i)}$ is not to be confused with X_i that signifies the stage duration distribution for the i^{th} stage.

rameters (e.g., λ for exponential distributions, or μ, σ for normal/log-normal distributions), is written as: $\theta_{MLE} = \arg \max_{\theta} P(X_{(1)} = x_{(1)}, X_{(2)} = x_{(2)}, \dots, X_{(r)} = x_{(r)}; \theta)$.

Unfortunately, it is computationally expensive to maximize the above likelihood expression in an online setting. Instead, we compute the maximum likelihood estimates of the parameters θ independently from each random variable $X_{(i)}$ and average the estimates together. While some internals of the estimation algorithm vary depending on the distribution type, the general idea remains the same. We present the details for log-normal and normal distributions. The maximum likelihood estimates for the order-statistics for the standard log-normal distribution are known, denoted by o_1, o_2, \dots, o_k henceforth.⁸ Since there are two parameters to estimate, μ , and σ , at least two outputs are required. Let t_1 and t_2 denote the arrival times for the first two responses. Then, we have $\ln t_1 = \hat{\mu} + \hat{\sigma} \ln o_1$, and $\ln t_2 = \hat{\mu} + \hat{\sigma} \ln o_2$. This gives us the first estimate of μ and σ . The i^{th} estimate comes from t_i and t_{i+1} and the final estimates are obtained by averaging individual estimates. The method for normal distribution is similar; the equations do not have a logarithm on either side.

4.3 Optimal Wait Duration

Once the underlying distribution is estimated, the next step is selection of the optimal value of the wait duration to maximize the quality (CALCULATEWAIT in Pseudocode 1, and Pseudocode 2). As before, we focus our attention only on the quality contributed by a single aggregator, since the contribution of different aggregators to the overall quality is independent of each other.

At a high level, the intuition is to model the expected gain and loss in qualities due to a small additional wait. We next formalize the gain and loss in quality. For ease of understanding, we present the analysis for a two-level tree (§4.3.1), before generalizing it to a n -level tree (§4.3.2).

4.3.1 Two-level tree

Consider an aggregator that has waited for t units of time and has not received all the outputs. A small additional wait of Δt can result in additional responses being collected by the aggregator.

Improvement in Quality: The probability that a process’s output is received by the aggregator in time $(t, t + \Delta t]$ is given by $a = (\phi_{X_1}(t + \Delta t) - \phi_{X_1}(t))$, where ϕ_{X_1} is the CDF of X_1 . The number of additional outputs from processes received in the Δt interval, then, is a binomial random variable with success probability a . The expected number of additional outputs received (given that the random variable is binomial) is then $k_1 a$,

⁸These are values that are available online or can be computed quite accurately using a simple simulation.

where k_1 is the maximum number of outputs (fanout) that an aggregator can collect. These additional processes add to the quality of the final response only if they reach the top-level aggregator in time whose probability is $b = \phi_{X_2}(D - (t + \Delta t))$. The expected gain due to these additional outputs is given by multiplying a with b :

$$k_1(\phi_{X_1}(t + \Delta t) - \phi_{X_1}(t)) \cdot \phi_{X_2}(D - (t + \Delta t)) \quad (1)$$

Since quality is the fraction of process outputs, the expected gain in quality is obtained by dividing the above expression by k_1 .

Reduction in quality: The additional wait of Δt , however, might lead to all the outputs of the aggregator (including the additional garnered ones) not being included in the final result. This happens if the aggregator itself misses its deadline and is ignored altogether. The expected number of outputs received till time t is $k_1 \frac{(\phi_{X_1}(t) - [\phi_{X_1}(t)]^{k_1})}{1 - [\phi_{X_1}(t)]^{k_1}}$ (Appendix C). The probability that the deadline is missed due to the additional waiting is $\phi_{X_2}(D - t) - \phi_{X_2}(D - (t + \Delta t))$. However, the above loss only occurs if all the outputs have not been collected by the aggregator, which happens with probability $(1 - [\phi_{X_1}(t)]^{k_1})$. Thus, the expected loss in process outputs is obtained by multiplying the above three expressions:

$$k_1(\phi_{X_1}(t) - [\phi_{X_1}(t)]^{k_1}) \cdot (\phi_{X_2}(D - t) - \phi_{X_2}(D - (t + \Delta t))) \quad (2)$$

Dividing the above expression by k_1 gives us the expected loss in quality.

4.3.2 Extension to n -level tree

Denote the number of levels in the aggregation tree to be n , the fanout of each level denoted by k_1, k_2, \dots, k_n , and stage duration distributions by X_1, X_2, \dots, X_n ; X_1 being the lowermost stage. Denote $q_n(D, X_1, k_1, X_2, k_2, \dots, X_n, k_n)$ (abbreviated as $q_n(D)$ whenever $X_1, k_1, \dots, X_n, k_n$ can be treated implicit) to be the maximum quality (in expectation) of this aggregation tree. The previous section formulated the gain and loss in q_2 .

To extend our formulation to more than two levels, we devise a recursive formulation by expressing the gain and loss in q_n in terms of q_{n-1} . The key observation that we make is that the *maximum quality achieved under a certain deadline, $q_n(D)$, is exactly the same as the maximum probability that a particular process' output reaches the root*. This happens only when each aggregator in the hierarchy selects the optimal wait-duration. For a single level tree, $q_1(D)$ is simply the probability of a process output reaching the (only) aggregator by the deadline D . Thus, $q_1(D, X_1, k_1) = \mathbb{P}[X_1 \leq D] = \phi_{X_1}(D)$.

Therefore, the changes to the expressions for gain and loss of quality are as follows.

```

procedure CALCULATEWAIT( $D, k_1$ , Distribution  $X_1$ )
  ▷ Also uses Distributions  $X_i$ 's and  $k_i$ 's for higher
  levels (global)

  double  $wait \leftarrow 0$                                 ▷ Wait Duration
  double  $q \leftarrow 0$ ;  $bestQ \leftarrow 0$                 ▷ Quality
  for double  $c = 0$ ;  $c \leq D$ ;  $c += \epsilon$  do
    ▷ Incremental search in steps of  $\epsilon$ 
    double  $G = \text{QUALITYGAIN}(c, X_1, k_1)$            ▷ Eqn. 3
    double  $L = \text{QUALITYLOSS}(c, X_1, k_1)$            ▷ Eqn. 4
     $q+ \leftarrow G - L$ 
    if  $q \geq bestQ$  then
       $bestQ \leftarrow q$ 
       $wait \leftarrow c$ 
  return  $wait$ 

```

Pseudocode 2: **Calculation of the optimal wait duration by balancing gain and loss. The optimal wait duration depends on the distributions X_1, X_2, \dots, X_n , deadline D and the fanouts k_1, k_2, \dots, k_n .**

Improvement in quality: The probability that the additional outputs collected in Δt reach the root is $q_{n-1}(D - (t + \Delta t), X_2, k_2, \dots, X_n, k_n)$, i.e., the maximum achievable quality for the $n - 1$ level tree beginning at X_2 (abbreviated as $q_{n-1}(D - (t + \Delta t))$ below). For a two level tree, this is $q_1(D - (t + \Delta t), X_2, k_2) = \phi_{X_2}(D - (t + \Delta t))$. Thus, the expression for gain in quality for a two-level tree, or q_2 , is $(\phi_{X_1}(t + \Delta t) - \phi_{X_1}(t)) \cdot q_1(D - (t + \Delta t), X_2, k_2) \triangleq$ Equation 1. Thus, the expected gain in quality for an n -level tree is:

$$(\phi_{X_1}(t + \Delta t) - \phi_{X_1}(t)) \cdot q_{n-1}(D - (t + \Delta t)) \quad (3)$$

Reduction in quality: To get the expression for n levels, we need to replace $\phi_{X_2}(\cdot)$ by $q_{n-1}(\cdot)$ in Equation 2 which gives:

$$(\phi_{X_1}(t) - [\phi_{X_1}(t)]^{k_1}) \cdot (q_{n-1}(D - t) - q_{n-1}(D - (t + \Delta t))) \quad (4)$$

The loss in q_2 is $(\phi_{X_1}(t) - [\phi_{X_1}(t)]^{k_1}) \cdot (q_1(D - t, X_2, k_2) - q_1(D - (t + \Delta t), X_2, k_2)) \triangleq$ Equation 2.

This recursive nature enables us to simply extend our algorithm to any number of levels.

4.3.3 Picking Wait Duration

Pseudocode 2 describes the algorithm for picking the optimal wait duration. Note that since the closed form solution is not known, we compute the wait duration by searching the space in small increments of ϵ . The net change in quality is the difference between the expressions in Equation 3 and Equation 4. We pick the value of wait duration which maximizes the quality. By keeping the value of ϵ to be small, we can reduce the discretization error. Note that while Pseudocode 2 provides a serial exploration of the space for wait duration, the

exploration is easily parallelizable, *i.e.*, we can perform the calculation for each value of ϵ independently. Further, one can simply precompute these wait-durations for recorded distributions.

5 Evaluation

We evaluate Cedar using an implementation over the Spark framework [15] and a simulator. We first explain the methodology of our evaluation before proceeding to present the results.

5.1 Methodology

Implementation: We implement Cedar’s algorithm over Spark [15]. Spark caches data in-memory allowing for fast interactive querying. For this, we first implement an aggregator that can do partial aggregation, *i.e.*, send results upstream after some timeout even when a subset of the lower level tasks have completed. Along with minor changes in the scheduler, we are able to run an entire partition aggregate workflow. Finally, we implement the baseline and Cedar’s algorithm in the aggregators to select appropriate wait-duration. The total code is ~ 300 LOC in Scala; but Cedar’s algorithm took < 50 LOC. We deploy Cedar on an EC2 cluster of 80 quad-core machines (320 slots to run processes).

Simulator: Our simulator mimics aggregation queries and can take as its input different fanout factors, deadlines, as well as distributions (both real-world distributions as well as synthetic). We use the simulator to evaluate Cedar’s sensitivity to fanout values (§5.4), and when there are multiple levels in the aggregation tree (§5.5).

Workloads: We simulate Cedar using production traces from Facebook’s Hadoop cluster [18], RTT values in Bing’s search values [6], task duration statistics from Bing’s production analytics cluster [18], and statistics from Google’s search cluster [10]. We also evaluate the effect of variances in the distributions by synthetically injecting them to the original traces. For the latter, we change the parameters of a log-normal approximation learned from the traces.

Primary Workload: We use the production traces from the Facebook cluster as our primary workload where we have exact durations of map and reduce tasks per job. For a particular job, process durations are given by the map tasks and aggregator durations are given by the reduce tasks. In this way, we are able to replay individual jobs. Since, we have perfect information of task durations, we are also able to dissect Cedar’s performance in detail (§5.3).

While this workload is not user-facing, we believe it to be representative of cluster variations. Regardless, Cedar’s algorithm is robust to different distributions as shown by a comprehensive evaluation on workloads based on both production and synthetic traces (§5.6 and

§5.7).

Topology: We use a two level hierarchy for all our experiments (except when experimenting with multiple levels). Unless otherwise specified, the fanout at the lower level is fixed at 50 (based on values in Bing’s cluster [6]) and the upper layer fanout is also set to be 50. For Spark results, we set the lower layer fanout to be 20 and upper layer fanout to be 16 giving us a total of 320 processes. We also analyze the sensitivity of Cedar’s gains to the fanout.

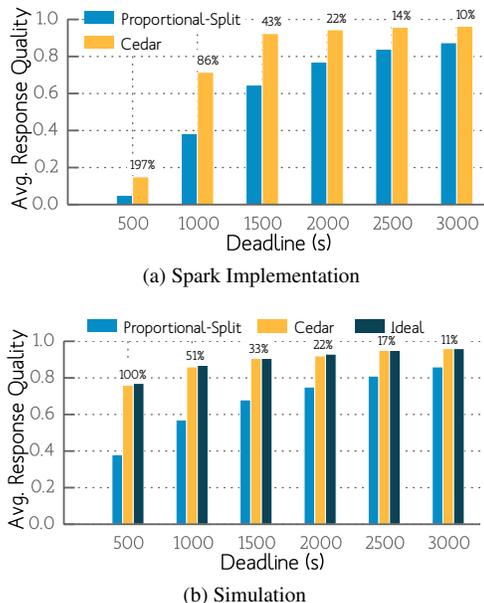


Figure 7: **Improvement in Response Quality.** X_1 is per Facebook’s Map distribution and X_2 is per Facebook’s Reduce distribution for different queries. The fanout at both levels is fixed at 50.

Metric: Our figure of merit is the increase in average response quality compared to the baseline of “Proportional-split” defined in §3. Proportional-split estimates the distribution in every level in the hierarchy from previous queries by fitting the best parameters. Therefore, if the quality of response achieved with our baseline and Cedar is $Quality_B$ and $Quality_C$ respectively, the improvement is defined as $100 \times \frac{Quality_C - Quality_B}{Quality_B}$.

We also report other percentile values, when appropriate, to show the spread in improvements. Further, we also compare Cedar’s performance to the “ideal” scheme described in §3. The ideal scheme is aware of distribution of process durations of all queries, and represents the maximum achievable improvement.

We start with the highlights of our results.

- Response quality improves by over 100% with Cedar compared to straw-man solutions. The absolute value of the quality goes to over 0.9. (§5.2)
- Online estimation of distribution parameters using

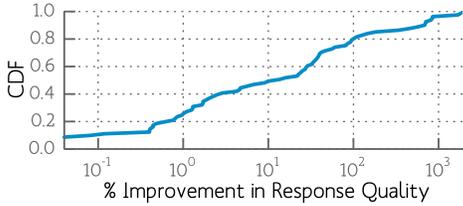


Figure 8: **CDF of Percentage improvement of individual queries. The deadline is set to 1000s. We only look at queries having > 5% quality in the baseline approach to prevent percentage improvement from being unreasonably high.**

order statistics results in less than 5% error even with very few samples. (§5.3)

- Cedar’s gains hold up for different fanouts and different distributions. (§5.4 and §5.7)
- Cedar’s importance only increases with number of stages in aggregation trees. (§5.5)
- Cedar’s algorithm is robust across different distributions. (§5.6 and §5.7)

5.2 Improvement in Response Quality

Figure 7 plots the average response quality achieved by Proportional-split as well as Cedar, along with the relative improvements for the Facebook workload.⁹ Our results show three key points. First, Cedar significantly improves the quality of the response over Proportional-split (the improvements lie between 10–197% in deployment and 11–100% in simulation). These results reinforce the extent to which variations in the distribution can affect response quality and the importance of picking the right wait duration. Second, while Cedar consistently pushes the quality to over 0.9 at deadlines > 1000s, the baseline *cannot* achieve a similar quality even at a humongous deadline of 3000s. Third, Cedar’s performance closely matches that of the ideal system that is aware of process distribution of the query beforehand (Figure 7b).

Figure 8 plots the distribution in improvements at the deadline of 1000s. 40% of the queries see their quality improve by over 50%. However, the bottom one-fifth of queries see little gains. This is primarily due to the long tail in the distribution of process durations in these queries, leaving little scope for improvement in quality regardless of the wait duration. Improvement in quality of these queries will occur only by specific techniques that reduce systemic and network contentions. Efforts to that end are focus of many current research projects, and Cedar’s algorithms will beneficially coexist with them.

5.3 Dissecting Cedar’s Learning

We next turn to dissecting Cedar’s performance to better understand the reasons behind the improvements. There

⁹We prune the trace to only consider jobs with > 2500 map tasks (for 2500 processes) and > 50 reduce tasks (for 50 aggregators).

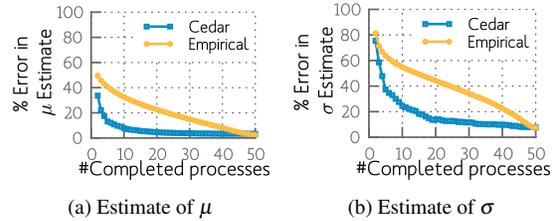


Figure 9: **Variation in % error in estimation of the μ and σ parameters of Facebook’s distribution (log-normal with $\mu = 4.77$ and $\sigma = 0.84$) with the number of responses that have arrived at the aggregator (maximum of 50). The baseline is the empirical estimates for μ and σ from the responses.**

are two factors in Cedar’s learning mechanism contributing to its gains—eliminating bias in received samples using order statistics (§5.3.1); and a simple yet accurate online learning algorithm (§5.3.2).

5.3.1 Estimation using Order Statistics

Recall from §4.2 that Cedar uses order statistics to estimate the mean μ and standard deviation σ of the distribution. This helps us to eradicate the error in its estimates despite being provided a biased sample of durations from only the early processes. We compare it with an “empirical” technique that estimates the mean and standard deviation directly from the available responses, and is hence susceptible to biased samples.

Figure 9 compares the error in Cedar’s estimation to the empirical technique, as the number of samples increases. Cedar’s estimation of μ is not only more accurate, the error also drops off to less than 5% when at least ten processes have completed. Error in estimation of σ is relatively higher ($\sim 20\%$), however it has a lesser effect on the wait duration. This is also evidenced by Cedar’s improvements closely matching an ideal scheme (Figure 7).

Regardless, Cedar’s improvements in response quality are 30–70% higher than the empirical technique (Figure 10), due to the use of order statistics in its learning.

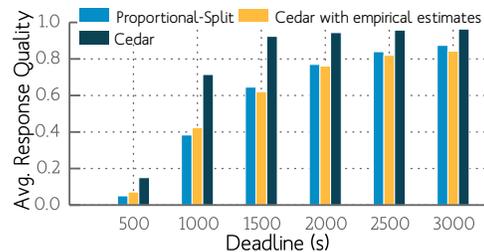


Figure 10: **Spark implementation results showing that Cedar’s learning algorithm provides significant benefits over using empirical estimates for the parameters.**

5.3.2 Importance of Online Learning

Cedar estimates the wait duration by learning the parameters of the distribution per query in an online fash-

ion. Such learning has significant impact on performance. To illustrate this, we consider the processes in the aggregation tree to be first operating at a lower load than Facebook’s map distribution, and use a $\text{log-normal}(2.77, 0.84)$ distribution to model X_1 (i.e., we keep σ to be as per the Facebook’s distribution but with lower μ). Figure 11 shows what happens when the load increases, and the distribution becomes the same as Facebook’s map distribution. If Cedar’s optimal wait-duration computation algorithm is used when the load is low, then the quality of the responses was $> 90\%$. However, if the same wait-duration (that was ideal previously) is used when the load increases then the quality of responses drops. Since Cedar learns the distribution in an online fashion, it is able to cope with such load fluctuations.

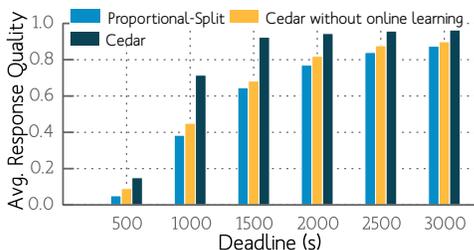


Figure 11: **Spark implementation results showing that Cedar copes well with load fluctuations that can increase (or decrease) mean process durations by learning the distributions in an online fashion.**

5.4 Effect of Fanout

While our experiments so far have assumed a fanout of 50 at both levels based on values from Bing’s cluster [6], we evaluate the performance of Cedar with differing fanout values using our trace-driven simulator.

Equal fanout at both levels. We vary the fan-out value of both the levels in the hierarchy and plot the results in Figure 12a. We observe that at lower values of fanout, Cedar’s gains are slightly lower. This is because at lower values of fanout, there are quadratically fewer processes and hence reduced variation between process durations. Therefore, the potential gains achievable by Cedar are slightly less. However, beyond a fanout of 25, Cedar’s estimation starts showing value with $\sim 50\%$ gain.

Different fanout across levels. We now compare the performance of Cedar for differing values of fanout in the two hierarchies. The fanout in the upper level of the hierarchy, k_2 , is set to 50 while the lower level’s fanout, k_1 , is varied between 5 and 50. Figure 12b plots the improvement in response quality with the ratio of k_1 to k_2 . Beyond a value of 0.2 for the ratio, the improvements stabilize and hover around 55%. Varying the ratio of k_1 over k_2 to over 1 does not change the trend in improvements.

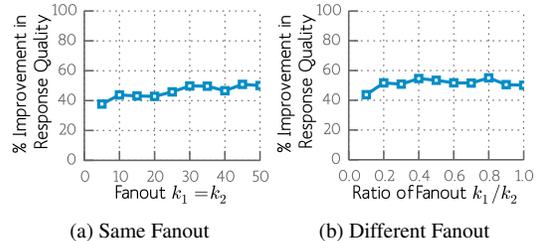


Figure 12: **Simulation results showing that Cedar’s gains hold up when the structure of the aggregation tree changes. In (a), we keep the fanout at both the levels. In (b), we choose different fanout for the lower-level while keeping the upper-level fanout at 50. The deadline is set to 1000s.**

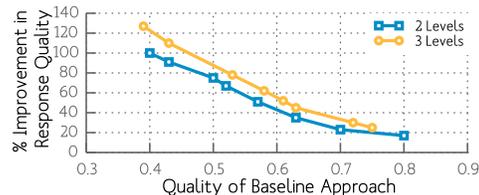


Figure 13: **Simulation results showing that Cedar performs even better when the number of levels in the aggregation tree increases.**

5.5 Multiple Stages

Given that Cedar’s formulation is recursive, it is directly applicable to aggregation trees with more than two stages in the hierarchy. To evaluate if Cedar’s gains hold up when the number of levels in the hierarchy increase, we consider a 3-stage aggregation tree. We model the lowest level using Facebook’s Map distribution and the upper two levels using Facebook’s Reduce distribution. Figure 13 compares the percentage improvement in response quality over the baseline (proportional-split) for a two-level and a three-level aggregation tree. Since, the three-level would require higher deadline values to achieve the same quality, we instead plot quality of the baseline approach on the x -axis to make a fair comparison. We observe that not only Cedar’s gains hold up, they provide greater improvements for higher number of stages. This is because Cedar near-optimally balances the deadline among the different stages which becomes more crucial as the number of stages increase.

5.6 Other Production Workloads

In this section, we consider a number of different setups based on other production workloads.

Interactive workload: The Hadoop workload at Facebook, while representative of performance variations encumbering large clusters, has really large process and aggregator durations. We consider a workload where the lower stage is modeled as per the Facebook’s map distribution (albeit expressed in ms) and the upper stage is modeled by the Google’s distribution (already in ms). Thus, this workload has higher variations in the lower

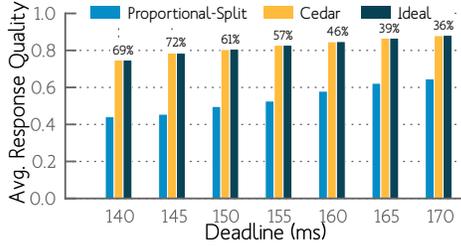


Figure 14: **Improvement in Response Quality.** X_1 is per Facebook’s map distribution and X_2 is per Google’s distribution. The fanout at both layers is 50.

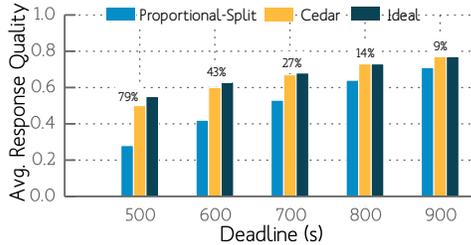


Figure 15: **Improvement in Response Quality.** X_1 is per Cosmos’s *extract* distribution and X_2 is per Cosmos’s *full-aggregate* distribution. The fanout at both levels is fixed at 50.

stage compared to the upper stage. We believe this to be representative of partition-aggregate workflows for the following two reasons. First, processes at the lower levels perform arbitrary user-defined functions and hence are susceptible to multiple local systemic contentions (as in the Facebook traces), while aggregators perform relatively standard functions and are more influenced by networking and scheduling factors (as in the Google and Bing traces). Second, variation in process durations is statistically expected to be higher than among aggregators because there are far more of them in a query. Since the Facebook distribution has much higher variation than the Google trace, our assumption helps match the statistical expectation. The deadline is varied from 140 – 170ms (based on quoted deadlines values for production search queries [5, 2]). Figure 14 plots the results. Cedar provides significant improvements over the baseline algorithm and manages to nearly match ideal performance even in this scenario.

Analytics cluster at Bing: We obtained statistics about the task duration values from an analytics cluster running in production at Bing. We run Cedar on this workload. The lower stage is modeled using the statistics from *extract* phase with a mean duration of 422s and standard deviation of 1006s; and the upper stage is modeled as per the *full-aggregate* phase with a mean duration of 193s and standard deviation of 92s. Despite the fact that Cedar’s online learning algorithm is not in play here due to the lack of task durations *per job*, Cedar provides considerable improvements in response quality as shown in

Figure 15 and comes close to the ideal scheme. We expect the improvements to only be higher if the per-job task durations were available.

Similar Distribution at both stages: We also used similar distribution for X_1 and X_2 , that are derived from each of Bing, Google and Facebook distributions. We evaluate Cedar for varying values of σ of X_1 , *i.e.*, the lower stage.¹⁰ The upshot is that Cedar’s performance continues to match the gains of an ideal scheme.

Bing’s Distribution: We consider the case where both the stages are distributed as per the Bing distribution (a log-normal fit with parameters, $\mu = 5.9$ and $\sigma = 1.25$, in μs). We are interested in the case when both levels have different amount of variabilities and thus, vary the σ parameter of the process duration distribution. We plot the % improvement over Proportional-split and also compare against the improvement of the ideal scheme in Figure 16a.

Google Distribution: We perform a similar experiment as above when both stages are distributed as per Google’s cluster (log-normal fit with parameters, $\mu = 2.94$ and $\sigma = 0.55$ in ms). Figure 16b shows how the performance gains (compared to Proportional-split) varies as the variability increases among process durations.

Facebook Distribution: Finally, we use the distribution from Facebook’s Hadoop cluster logs for X_2 and samples from a log-normal fit for these map-task durations for X_1 , studying the performance gains as one induces more variance in the first stage in Figure 16c.

5.7 Other Distribution Types

Since all our traces fit the log-normal distribution, our results thus far, were based on that. To demonstrate that Cedar is agnostic to the type of distribution, we evaluate its performance with the Gaussian distribution. The experiment uses a two-level tree with process durations distributed normally with mean 40ms at both the levels; the standard deviation being 10ms and 80ms for the top and bottom levels respectively (keeping variance at bottom level higher than above levels). As Figure 17 shows, while the percentage improvements are smaller than the log-normal cases, Cedar achieves quite high absolute values of quality. This is expected given that normal distributions are not heavy tailed.

6 Discussion and Related Work

Straggler mitigation techniques work to reduce the variability in task durations [8, 9, 7, 5, 21, 6]. First, Cedar can complement these mitigation techniques, since stragglers still occur *despite* them (as seen in our traces). Second, by virtue of being *reactive*, straggler mitigation techniques fail to work effectively when process durations

¹⁰The results for varying σ of X_2 look similar and we omit them in the interest of space.

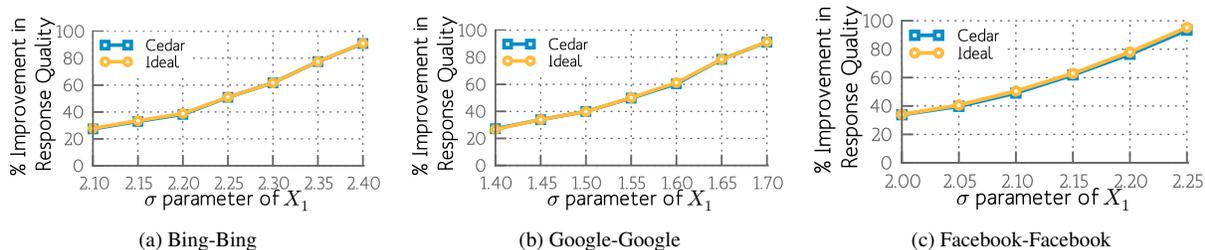


Figure 16: **Same Distributions: Percentage Improvement in Response Quality as the σ parameter of X_1 is varied.** μ parameter of log-normal distributions X_1 and X_2 and σ parameter of X_2 are obtained from (a) Bing, (b) Google, and (c), Facebook distributions.

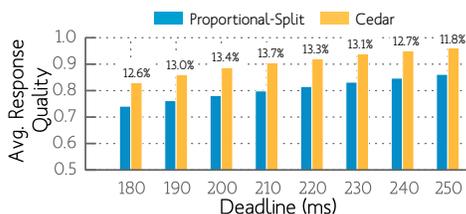


Figure 17: **Cedar’s performance with the Gaussian distribution.** The percentages on the bars denote the improvement that Cedar provides over Proportional-Split.

are sub-second, as for interactive queries [9], whereas Cedar still works well. Third, while straggler mitigation techniques attempt to remove variance from within a query (or a job), some queries are inherently more expensive (computationally or otherwise) than other. Cedar tailors a query-specific wait duration to improve application performance.

In **approximate analytics**, recent work, GRASS [3], has looked at mitigating stragglers in approximate-analytics. Unlike GRASS, Cedar’s benefits hold whether or not a stage is single-wave (the common case for partition-aggregate style workloads, e.g., web-search) or multi-wave. This is because while GRASS primarily focusses on the question of which task to schedule when a slot frees up within one stage of a job, Cedar focuses on optimizing the wait time between stages. This leads to two important differences. First, GRASS considers each stage of a job independently; Cedar optimizes the stages jointly (by optimizing wait-durations at aggregators). Second, GRASS’s scheduling benefits only “multi-wave” stages in a job – i.e., stages with more tasks than slots available. Cedar treats the question of when and how tasks should be scheduled as orthogonal. Thus, in summary, the two are complementary.

Deadline-aware scheduling has garnered significant attention recently, both in systems [22, 23, 24] and networking [25, 2, 11, 26] communities. The networking community has focussed on meeting flow deadlines such that the *application throughput* (analogous to response quality) is maximized. However, such approaches aim to improve the performance of a single level. Cedar’s approach is end-to-end, in that it aims to maximize the

final response quality without worrying about individual stages. The systems community has also been looking at providing job SLOs [22], but the focus has been on jobs that require exact results which do not trade-off quality of the response with its latency. Kwiken [24] improves performance of request-response workflows using a variety of techniques including request reissues, catching-up on laggards, and trading off accuracy for responsiveness. Cedar’s approach is closest to the last technique in that it solves the dual problem of maximizing accuracy under a desired responsiveness. Cedar differs as it considers the entire partition-aggregate workflow in a holistic way. Further, Cedar’s online learning algorithm using order-statistics can aid in determining reissue budget across stages in a better way.

Consider the **alternate system model** of running an approximate-query system where the deadline is set such that $x\%$ of the process outputs are collected at the root. This imposes a threshold on response quality instead of its latency. Since Cedar’s algorithm is solving the dual problem, it can be applied to such systems as well, i.e., Cedar can provide the same quality threshold ($x\%$) at a lower deadline value thereby improving query’s response time.

7 Conclusion

We formalize the dilemma that an aggregator faces whilst deciding whether it should wait for additional time in the hope of getting new process outputs, or to stop in order to meet the upper-level deadline. We show that wait-time duration selection has great potential (over 100%) to improve the quality of responses within tight time budgets. Our solution Cedar, builds upon (i) an algorithm to perform online estimation of stage-duration distribution parameters; and (ii), a theoretically optimal algorithm to maximize the expected quality given the distributions. We show that Cedar achieves near-optimal improvements in response qualities under a variety of distributions, most notably so when there is high variability in the system.

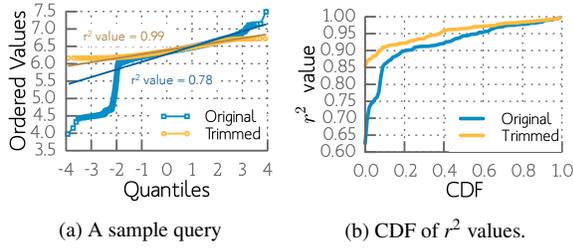


Figure 18: **Goodness of log-normal fit for Facebook Data.** (a) shows that the part of the distribution that Cedar is interested in is much more accurately modeled by a log-normal fit compared to the original distribution. (b) shows that the coefficient of determination (r^2 value) is greater than 0.9 for more than 90% of the jobs

A Weighted Process Outputs

Consider the case when each process output is not equally important. Denote $w : \mathbb{O} \rightarrow [0, 1]$ to be the quality function that assigns a weight to $x \in \mathbb{O}$ between $[0, 1]$. The constraint is that $\sum_{x \in \mathbb{O}} w(x) = 1$. Given that Cedar maximizes the response quality in expectation the analysis in §4.3 holds true. For example, the additional quality garnered due to an additional waiting of Δt can be written as $\mathbb{E}_q = \sum_{x \in \mathbb{O}} w(x) \cdot p_x$ where p_x denotes the probability that the output comes in the interval $(t, t + \Delta t]$. Clearly, p_x is still equal to $\phi_{X_1}(t + \Delta t) - \phi_{X_1}(t)$ and is independent of x . Therefore, $\mathbb{E}_q = p_x \sum_{x \in \mathbb{O}} w(x) = p_x$. The analysis for other terms is also similar to this and thus, the algorithm holds.

B Goodness of Fit

We characterize the goodness of fit for our data. As stated before, we checked against a variety of distributions and log-normal distribution gave us the best fit. This section aims to characterize how well did the data fit the log-normal distribution. We took logarithm of the durations to test the normality of data using a quantile-quantile plot [27]; being interested the r^2 -values (coefficient of determination). A linear quantile-quantile plot illustrates a linear relationship between the data and the model. For the Facebook data, we observed that a large number of queries had low r^2 -values, the outliers mostly being in the initial values and towards the tail. For the purposes of Cedar, the initial few outputs will always be aggregated even with a non-optimal value of wait-duration. Further, given the long-tail behavior of the data, higher percentiles, say 99% or 99.5%, will be stragglers which will not be aggregated irrespective of any optimization of the wait-duration. We remove the first 10% and the last 0.5% of the data and observe that the fit improves considerably. As an example, consider Figure 18a which shows a job for which the coefficient of determination goes up from 0.77 to 0.99 when the dataset is trimmed slightly. The CDF for the 100 jobs sampled

from the set is plotted in Figure 18b both for the original as well as trimmed durations. The coefficient of determination is greater than 0.9 for more than 90% of the jobs.

Bing’s data had a heavier and longer tail compared to Facebook’s data (considered per query), *e.g.*, the 99th percentile was 42× the median value. Even the 95th percentile was 15× the median value. Thus, we truncated the distribution till 95th percentile giving us a coefficient of determination of 0.98 (compared to 0.94 when the distribution was truncated to 99th percentile).

C Expected Number of Process Outputs

Consider one of the bottom-most aggregators in the aggregation tree with X_1 as the the distribution of process durations and k downstream processes. We explain the calculation for the number of responses that this aggregator has received till time t , $N(t)$. The conditional expectation that we want is $\mathbb{E}[N(t)|N(t) \neq k]$, since the decision on waiting more arises only when more outputs have to arrive. Denote the probability of receiving a particular output by time t by $p = \phi_{X_1}(t)$. The probability of not receiving all outputs by time t is $\mathbb{P}[N(t) \neq k] = 1 - p^k$. The conditional probability of receiving r process outputs by time t given $N(t) \neq k$ is $\mathbb{P}[N(t) = r|N(t) \neq k] = \frac{\binom{k}{r} p^r (1-p)^{k-r}}{1-p^k}$ for $r < k$, and 0 for $r = k$. The conditional expectation can now be derived as follows:

$$\begin{aligned}
 \mathbb{E}[N(t)|N(t) \neq k] &= \sum_{r=1}^{k-1} r \mathbb{P}[N(t) = r|N(t) \neq k] \\
 &= \sum_{r=1}^{k-1} r \frac{\binom{k}{r} p^r (1-p)^{k-r}}{1-p^k} \\
 &= k \frac{p}{1-p^k} \sum_{r=1}^{k-1} \binom{k-1}{r-1} p^{r-1} (1-p)^{k-r} \\
 &= k \frac{p}{1-p^k} \sum_{m=0}^{k-2} \binom{k-1}{m} p^m (1-p)^{k-1-m} \\
 &= k \frac{p}{1-p^k} \left[\left(\sum_{m=0}^{k-1} \binom{k-1}{m} p^m (1-p)^{k-1-m} \right) - p^{k-1} \right] \\
 &= k \frac{p}{1-p^k} \left[1 - p^{k-1} \right] \\
 &= k \frac{\phi_{X_1}(t) - [\phi_{X_1}(t)]^k}{1 - [\phi_{X_1}(t)]^k}
 \end{aligned}$$

References

- [1] Sameer Agarwal, Anand P. Iyer, Aurojit Panda, Samuel Madden, Barzan Mozafari, and Ion Stoica. Blink and it's done: interactive queries on very large data. *Proc. VLDB Endow.*, 2012.
- [2] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: meeting deadlines in datacenter networks. *ACM SIGCOMM*, 2011.
- [3] G. Ananthanarayanan, M. C. Hung, X. Ren, I. Stoica, A. Wierman, M. Yu. Grass: Trimming stragglers in approximation analytics. In *NSDI*, 2014.
- [4] J. Dean. Achieving Rapid Response Times in Large Online Services. In *Berkeley AMPLab Cloud Seminar*, 2012.
- [5] David Zats, Tathagata Das, Prashanth Mohan, Dhruva Borthakur, and Randy Katz. Detail: reducing the flow completion time tail in datacenter networks. *ACM SIGCOMM*, 2012.
- [6] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *ACM SIGCOMM*, 2010.
- [7] Ganesh Ananthanarayanan, Srikanth Kandula, Albert Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in map-reduce clusters using mantri. *Usenix OSDI*, 2010.
- [8] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. *Usenix OSDI*, 2008.
- [9] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. *Usenix NSDI*, 2013.
- [10] Jeffrey Dean and Luiz André Barroso. The Tail at Scale. *Commun. ACM*, 2013.
- [11] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM*, 2012.
- [12] Herbert A. David and H. N. Nagarajan. *Order Statistics, 3rd Edition*. Wiley, 2003.
- [13] Jeff Rothschild. High Performance at Massive Scale Lessons learned at Facebook. <http://video-jsoc.ucsd.edu/asx/JeffRothschildFacebook.asx>.
- [14] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken. The Nature of Datacenter Traffic: Measurements and Analysis. *ACM IMC*, 2009.
- [15] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. *Usenix NSDI*, 2012.
- [16] Ron Kohavi and Roger Longbotham. Online experiments: Lessons learned. *IEEE Computer*, 2007.
- [17] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: interactive analysis of web-scale datasets. *Proc. VLDB Endow.*, 2010.
- [18] Ganesh Ananthanarayanan, Ali Ghodsi, Andrew Wang, Dhruva Borthakur, Srikanth Kandula, Scott Shenker, and Ion Stoica. Pacman: coordinated memory caching for parallel jobs. *Usenix NSDI*, 2012.
- [19] rrisk: Risk modelling and auto-reporting in r. http://www.bfr.bund.de/en/rrisk_risk_modelling_and_auto_reporting_in_r-52162.html.
- [20] Allen B. Downey. Lognormal and pareto distributions in the internet. *Comput. Commun.*, 28(7):790–801, May 2005.
- [21] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: trading a little bandwidth for ultra-low latency in the data center. *Usenix NSDI*, 2012.
- [22] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: guaranteed job latency in data parallel clusters. *ACM EuroSys*, 2012.
- [23] Virajith Jalaparti, Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Bridging the tenant-provider gap in cloud services. *ACM SoCC*, 2012.
- [24] Virajith Jalaparti, Peter Bodik, Srikanth Kandula, Ishai Menache, Mikhail Rybalkin, and Chenyu Yan. Speeding up distributed request-response workflows. In *SIGCOMM*, 2013.
- [25] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM*, 2012.
- [26] Mohammad Alizadeh, Shuang Yang, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Deconstructing datacenter packet transport. *ACM HotNets-XI*, 2012.
- [27] Wikipedia. QQ plot — Wikipedia, the free encyclopedia, 2014.