Design Insights for MapReduce from Diverse Production Workloads

Yanpei Chen
Sara Alspaugh
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-17
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-17.html

January 25, 2012

Acknowledgement

The authors are grateful for the feedback from our colleagues at UC Berkeley AMP Lab, Cloudera, and Facebook. We especially appreciate the inputs from Anthony Joseph, David Zats, Matei Zaharia, Jolly Chen, Todd Lipcon, Aaron T. Myers, John Wilkes, and Srikanth Kandula. This research is supported in part by AMP Lab (https://amplab.cs.berkeley.edu/sponsors/), and the DARPA- and SRC-funded MuSyC FCRP Multiscale Systems Center.

Design Insights for MapReduce from Diverse Production Workloads

Yanpei Chen, Sara Alspaugh, Randy Katz
University of California, Berkeley

Abstract

In this paper, we analyze seven MapReduce workload traces from production clusters at Facebook and at Cloudera customers in e-commerce, telecommunications, media, and retail. Cumulatively, these traces comprise over a year's worth of data logged from over 5000 machines, and contain over two million jobs that perform 1.6 exabytes of I/O. Key observations include input data forms up to 77% of all bytes, 90% of jobs access KB to GB sized files that make up less than 16% of stored bytes, up to 60% of jobs re-access data that has been touched within the past 6 hours, peak-to-median job submission rates are 9:1 or greater, an average of 68% of all compute time is spent in map, task-seconds-per-byte is a key metric for balancing compute and data bandwidth, task durations range from seconds to hours, and five out of seven workloads contain map-only jobs. We have also deployed a public workload repository with workload replay tools so that the researchers can systematically assess design priorities and compare performance across diverse MapReduce workloads.

1 Introduction

The MapReduce computing paradigm is gaining widespread adoption across diverse industries as a platform for large-scale data analysis, accelerated by open-source implementations such as Apache Hadoop. The scale of such systems and the depth of their software stacks create complex challenges for the design and operation of MapReduce clusters. A number of recent studies looked at MapReduce style systems within a handful of large technology companies [19, 8, 39, 33, 30, 10]. The stand-alone data sets used in these means that the associated design insights may not generalize across use cases, an important concern given the emergence of MapReduce users in different industries [5]. Consequently, there is a need to develop systematic knowledge of MapReduce behavior at both established users within technology companies, and at recent adopters in other industries.

In this paper, we analyze seven MapReduce workload traces from production clusters at Facebook and at Cloudera's customers in e-commerce, telecommunications, media, and retail. Cumulatively, these traces comprise over a year's worth of data, covering over two million jobs that moved approximately 1.6 exabytes spread over 5000 machines (Table 3). They are collected using standard tools in Hadoop, and facilitate comparison both across workloads and over time.

Our analysis reveals a range of workload behavior. Key similarities between workloads include input data forms up to 77% of all bytes, 90% of jobs access KB to GB sized files that make up less than 16% of stored bytes, up to 60% of jobs re-access data that has been touched within the past 6 hours, peak-to-median job submission rates are 9:1 or greater, an average of 68% of all compute time is spend in map, task durations range from seconds to hours, and five out of seven workloads contain map-only jobs. Key differences include the frequency of data re-access, the burstiness of the workload, the balance between computation to data bandwidth, the analytical frameworks used on top of MapReduce, and the multi-dimension descriptions of common job categories. These results provide empirical evidence to inform the design and evaluation of schedulers, caches, optimizers, and networks, in addition to revealing insights for cluster provisioning, benchmarking, and workload monitoring.

Table 1 summarizes our findings and serves as a roadmap for the rest of the paper. Our methodology extends [19, 18, 17], and organizes the analysis according to three conceptual aspects of a MapReduce workload: data, temporal, and compute patterns. Section 3 looks at data patterns. This includes aggregate bytes in the MapReduce input, shuffle, and output stages, the distribution of per-job data sizes, and the per-file access frequencies and intervals. Section 4 focuses on temporal patterns. It analyzes variation over time across multiple workload dimensions, quantifies burstiness, and extracts temporal correlations between different workload dimensions. Section 5 examines computate patterns. It analyzes the balance between aggregate compute and data size, the distribution of task sizes, and the breakdown of common job categories both by job names and by multi-dimensional job behavior. Our contributions are:

- Analysis of seven MapReduce production workloads from five industries totaling over two million jobs,
- Derivation of design and operational insights, and
- Methodology of analysis and the deployment of a public workload repository with workload replay tools.

We invite other MapReduce researchers and users to add to our workload repository and derive additional insights

Section	Observations	Implications/Interpretations
3.1	Input data forms 51-77% of all bytes, shuffle 8-35%, output 8-23%.	Need to re-assess sole focus on shuffle-like traffic for datacenter networks.
3.2	The majority of job input sizes are MB-GB.	TB-scale benchmarks such as TeraSort are not representative.
3.2	For the Facebook workload, over a year, input and shuffle sizes increase by over 1000× but output size decreases by roughly 10×.	Growing customers and raw data size over time, distilled into possibly the same set of metrics.
3.3	For all workloads, data accesses follow the same Zipf distribution.	Tiered storage is beneficial. Uncertain cause for the common patterns.
3.3	90% of jobs access small files that make up 1-16% of stored bytes.	Cache data if file size is below a threshold.
3.3	Up to 78% of jobs read input data that is recently read or written by other jobs. 75% of re-accesses occur within 6 hrs.	LRU or threshold-based cache eviction viable. Tradeoffs between eviction policies need further investigation.
4.1	There is a high amount of noise in job submit patterns in all workloads.	Online prediction of data and computation needs will be challenging.
4.1	Daily diurnal pattern evident in some workloads.	Human driven interactive analysis, and/or daily automated computation.
4.2	Workloads are bursty, with peak to median hourly job submit rates of 9:1 or greater.	Scheduling and placement optimizations essential under high load. Increase utilization or conserve energy during inherent workload troughs.
4.2	For Facebook, over a year, peak to median job submit rates decrease from 31:1 to 9:1, while more internal organizations use MapReduce.	Multiplexing many workloads help smooth out bustiness. However, utilization remains low.
4.2	Workload intensity is multi-dimensional (jobs, I/O, task-times, etc.).	Need multi-dim. metrics for burstiness and other time series properties.
4.3	Avg. temporal correlation between job submit & data size is 0.21; for job submit & compute time it is 0.14; for data size & compute time it is 0.62.	Schedulers need to consider metrics beyond number of active jobs. MapReduce workloads remain data- rather than compute-centric.
5.1	Workloads on avg. spend 68% of time in map, 32% of time in reduce.	Optimizing read latency/locality should be a priority.
5.1	Most workloads have task-seconds per byte of 1×10^{-7} to 7×10^{-7}; one workload has task-seconds per byte of 9×10^{-4}.	Build balanced systems for each workload according to task-seconds per byte. Develop benchmarks to probe a given value of task-seconds per byte.
5.2	For all workloads, task durations range from seconds to hours.	Need mechanisms to choose uniform task sizes across jobs.
5.3.1	<10 job name types make up >60% of all bytes or all compute time.	Per-job-type prediction and manual tuning are possible.
5.3.1	All workloads are dominated by a 2-3 frameworks.	Meta-schedulers need to multiplex only a few frameworks.
5.3.2	Jobs touching <10GB of total data make up >92% of all jobs, and often have <10 tasks or even a single tasks.	Schedulers should design for small jobs. Re-assess the priority placed on addressing task stragglers.
5.3.2	Five out of seven workloads contain map-only jobs.	Not all jobs benefit from network optimizations for shuffle.
5.3.2	Common job types change significantly over a year.	Periodic re-tuning and re-optimizations is necessary.

Table 1: Summary of observations from the workload, and associated design implications and interpretations.

using the data in this paper. We hope that such public data will allow researchers and cluster operators to better understand and optimize MapReduce systems.

2 Background

This section reviews key previous work on workload characterization in general and highlight our advances. In addition, we describe the traces we used.

2.1 Prior Work

The desire for thorough system measurement predates the rise of MapReduce. Workload characterization studies have been invaluable in helping designers identify problems, analyze causes, and evaluate solutions. Table 2 summarizes some studies founded on the understanding realistic system behavior. They fall into the categories of network systems [29, 34, 36, 16, 24, 8], storage systems [35, 16, 32, 13, 19, 10], and large scale data centers running computation paradigms including MapReduce [8, 39, 33, 10, 30, 11]. Network and storage subsystems are key components for MapReduce. This paper builds on these studies.

A striking trend emerges when we order the studies by trace date. In the late 1980s and early 1990s, measurement studies often capture system behavior for only one setting [35, 29]. The stand-alone nature of these studies are due to the emerging measurement tools that created considerable logistical and analytical challenges at the time. Studies in the 1990s and early 2000s often traced the same system under multiple use cases [34, 36, 16, 32, 13, 24]. The increased generality likely comes from a combination of improved measurement tools, wide adoption of certain systems, and better appreciation of what good system measurement enables.

The trend reversed in recent years, with stand-alone studies becoming common again [19, 8, 39, 33, 10, 30]. This is likely due to the tremendous scale of the systems of interest. Only a few organizations can afford systems of thousands or even millions of machines. Concurrently, the vast improvement in measurement tools create an over-abundance of data, presenting new challenges to derive useful insights from the deluge of trace data.

These technology trends create the pressing need to generalize beyond the initial point studies. As MapReduce use diversifies to many industries, system designers need to optimize for common behavior [11], in addition to improving the particulars of individual use cases.

Some studies amplified their breadth by working with ISPs [36, 24] or enterprise storage vendors [13], i.e., intermediaries who interact with a large number of end customers. The emergence of enterprise MapReduce vendors present us with similar opportunities to generalize beyond single-point MapReduce studies.

2.2 Workload Traces Overview

We analyze seven workloads from various Hadoop deployments. All seven come from clusters that support business critical processes. Five are workloads from Cloudera's enterprise customers in e-commerce, telecommunications, media, and retail. Two others are Facebook workloads on the same cluster across two dif-

Study	Traced system	Trace date	Companies	Industries	Findings & contributions
Ousterhout *et al.* [35]	BSD file system	1985	1	1	Various findings re bandwidth & access patterns
Leland *et al.* [29]	Ethernet packets	1989-92	1	1	Ethernet traffic exhibits self-similarity
Mogul [34]	HTTP requests	1994	2	2	Persistent connections are vital for HTTP
Paxson [36]	Internet traffic	1994-95	35	>3	Various findings re packet loss, re-ordering, & delay
Breslau *et al.* [16]	Web caches	1996-98	6	2	Web caches see various Zipf access patterns
Mesnier *et al.* [32]	NFS	2001-04	2	1	Predict file properties based on name and attributes
Bairavasundaram *et al.* [13]	Enterprise storage	2004-08	100s	Many	Quantifies silent data corruption in reliable storage
Feamster *et al.* [24]	BGP configurations	2005	17	N/A	Discovers BGP misconfigurations by static analysis
Chen *et al.* [19]	Enterprise storage	2007	1	1	Various findings re placement, consolidation, caching
Alizadeh *et al.* [8]	Datacenter TCP traffic	2009	1	1	Use explicit congestion notification to improve TCP
Thereska *et al.* [39]	Storage for web apps	2009	1	1	Save energy by powering down data replicas
Mishra *et al.* [33]	Web apps backend	2009	1	1	Task classification for capacity planning & scheduling
Anathanarayanan *et al.* [10]	Web search (Dryad)	2009	1	1	Alleviate hotspots by pre-emptive data replication
Meisner *et al.* [30]	Web search	2010	1	1	Interactive latency complicates hardware power mngmt.
Anathanarayanan *et al.* [11]	Hadoop and Dryad	2009-11	3	1	Cache data based on heavy-tail access patterns

Table 2: Summary of prior work. The list includes network system studies [29, 34, 36, 16, 24, 8], storage system studies [35, 16, 32, 13, 19, 10], and studies on data centers running large scale computation paradigms including MapReduce [8, 39, 33, 10, 30, 11]. Note that studies in the 1990s and early 2000s have good breadth, while studies in the late 2000s returned to being stand-alone.

Trace	Machines	Length	Date	Jobs	Bytes moved
CC-a	<100	1 month	2011	5759	80 TB
CC-b	300	9 days	2011	22974	600 TB
CC-c	700	1 month	2011	21030	18 PB
CC-d	400-500	2+ months	2011	13283	8 PB
CC-e	100	9 days	2011	10790	590 TB
FB-2009	600	6 months	2009	1129193	9.4 PB
FB-2010	3000	1.5 months	2010	1169184	1.5 EB
Total	>5000	≈ 1 year	-	2372213	1.6 EB

Table 3: Summary of traces. CC is short for "Cloudera Customer". FB is short for "Facebook". Bytes touched is computed by sum of input, shuffle, and output data sizes for all jobs.

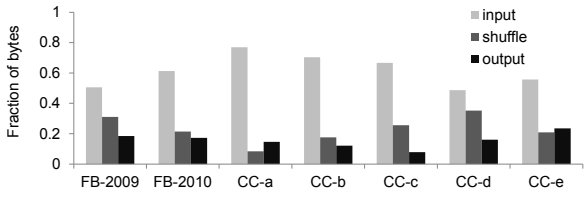

Figure 1: Aggregate fraction of all bytes that come from input, shuffle, or output of each workload.

ferent years. These workloads offer a rare opportunity to survey Hadoop use cases across several technology and traditional industries (Cloudera customers), and track the growth of a leading Hadoop deployment (Facebook).

Table 3 gives some detail about these workloads. The trace lengths are limited by the logistical challenges of shipping trace data for offsite analysis. The Cloudera customer workloads have raw logs approaching 100GB, requiring us to set up specialized file transfer tools. Transferring raw logs is infeasible for the Facebook workloads, requiring us to query Facebook's internal monitoring tools. Combined, the workloads contain over a year's worth of trace data, covering a non-trivial amount of jobs and bytes processed by the clusters.

The data is logged by standard tools in Hadoop; no additional tracing tools were necessary. The workload traces contain per-job statistics for job ID (numerical key), job name (string), input/shuffle/output data sizes (bytes), duration, submit time, map/reduce task time (slot-seconds), map/reduce task counts, and input/output file paths. We call each of these characteristic a numerical *dimension* of a job. Some traces have some data dimensions unavailable.

We obtained the Cloudera traces by doing a time-range selection of per-job Hadoop history logs based on the file timestamp. The Facebook traces come from a similar query on Facebook's internal log database. The traces reflect no logging interruptions, except for the cluster in CC-d, which was taken offline several times due to operational reasons. There are some inaccuracies at trace start and termination, due to incomplete jobs at the trace boundaries. The length of our traces far exceeds the typical job length on these systems, leading to negligible errors. To capture weekly behavior for CC-b and CC-e, we intentionally queried for 9 days of data to allow for inaccuracies at trace boundaries.

3 Data Access Patterns

Data movement is a key function of these clusters, so understanding data access patterns is crucial. This section looks at data patterns in aggregate (§ 3.1), by jobs (§ 3.2), and per-file (§ 3.3).

3.1 Aggregate input/shuffle/output sizes

Figure 1 shows the aggregate input, shuffle, and output bytes. These statistics reflect I/O bytes seen from the MapReduce API. Different MapReduce environments lead to two interpretations with regard to actual bytes moved in hardware.

If we assume that task placement is random, and locality is negligible for all three input, shuffle, and output stages, then all three MapReduce data movement stages involve network traffic. Further, because task placement

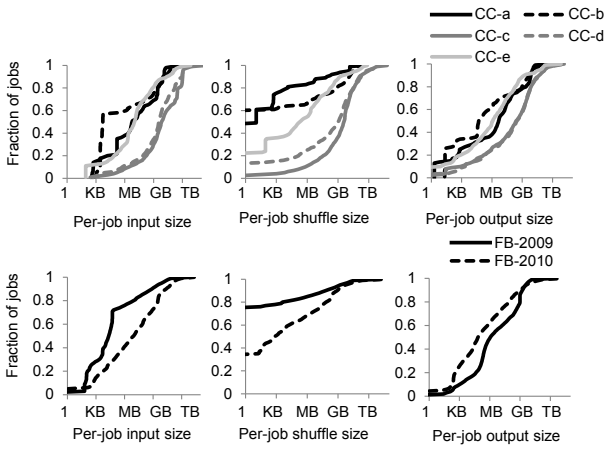

Figure 2: Data size for each workload. Showing input, shuffle, and output size per job.

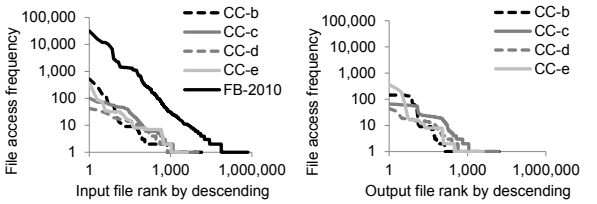

Figure 3: Log-log file access frequency vs. rank. Showing Zipf distribution of same shape (slope) for all workloads.

is random, the aggregate traffic looks like N-to-N shuffle traffic for all three stages. Under these assumptions, recent research correctly optimize for N-to-N traffic patterns for datacenter networks [7, 8, 25, 20].

However, the default behavior in Hadoop is to attempt to place map tasks for increased locality of input data. Hadoop also tries to combine or compress map outputs and optimize placement for reduce tasks to increase rack locality for shuffle. By default, for every API output block, HDFS stores one copy locally, another within-rack, and a third cross-rack. Under these assumptions, data movement would be dominated by input reads, with read locality optimizations being worthwhile [41]. Further, while inter-rack traffic from shuffle can be decreased by reduce task placement, HDFS output by design produces cross-rack replication traffic, which has yet to be optimized.

Facebook uses HDFS RAID, which employs Reed-Solomon erasure codes to tolerate 4 missing blocks with 1.4× storage cost [15, 37]. Parity blocks are placed in a non-random fashion. Combined with efforts to improve locality, the design creates another environment in which we need to reassess optimization priority between MapReduce API input, shuffle, and output.

3.2 Per-job data sizes

Figure 2 shows the distribution of per-job input, shuffle, and output data sizes for each workload. The median per-job input, shuffle, and output size respective differ by 6, 8, and 4 orders of magnitude. Most jobs have input, shuffle, and output sizes in the MB to GB range. Thus, benchmarks of TB and above [6, 4] captures only a narrow set of input, shuffle, and output patterns.

From 2009 to 2010, the Facebook workloads' per-job input and shuffle size distributions shift right (become larger) by several orders of magnitude, while the per-job output size distribution shifts left (becomes smaller). Raw and intermediate data sets have grown while the final computation results have become smaller. One possible explanation is that Facebook's customer base (raw data) has grown, while the final metrics (output) to drive business decisions have remained the same.

3.3 Access frequency and intervals

This section analyzes HDFS file access frequency and intervals based on hashed file path names. The `FB-2009` and `CC-a` traces do not contain path names, and the `FB-2010` trace contains path names for input only.

Figure 3 shows the distribution of HDFS file access frequency, sorted by rank according to non-decreasing frequency. Note that the distributions are graphed on log-log axes, and form approximate straight lines. This indicates that the file accesses follow a Zipf distribution.

The generalized Zipf distribution has the form in Equation 1, where $f(r;\alpha,N)$ is the access frequency of r^{th} ranked file, N is the total size of the distribution, i.e., number of unique files in the workload, and α is the shape parameter of the distribution. Also, $H_{N,\alpha}$ is the N^{th} generalized harmonic number. Figure 3 graphs $log(f(r;\alpha,N))$ against $log(r)$. Thus, a linear log-log graph indicates a distribution of the form in Equation 1, with $N/H_{N,\alpha}$ being a vertical shift given by the size of the distribution, and α reflects the slope of the line.

$$f(r;\alpha,N) = \frac{N}{r^{\alpha}H_{N,\alpha}} \quad H_{N,\alpha} = \sum_{k=1}^{N} \frac{1}{k^{\alpha}} \qquad (1)$$

Figure 3 indicates that few files account for a very high number of accesses. Thus, *any* data caching policy that includes those files will bring considerable benefit.

Further, the slope, i.e., α parameter of the distributions are all approximately 5/6, across workloads and for both inputs and outputs. Thus, file access patterns are Zipf distributions of the same shape. Figure 3 suggests the existence of common computation needs that leads to the same file access behavior across different industries.

The above observations indicate only that caching helps. If there is no correlation between file sizes and

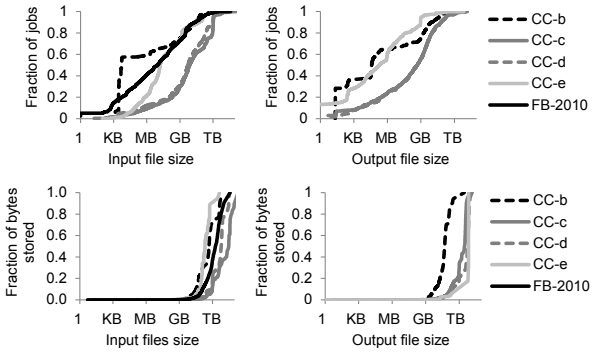

Figure 4: Access patterns vs. input/output file size. Showing cummulative fraction of jobs with input/output files of a certain size (top) and cummulative fraction of all stored bytes from input/output files of a certain size (bottom).

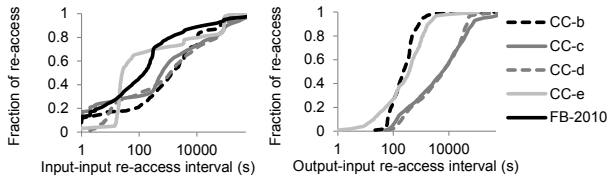

Figure 5: Data re-accesses intervals. Showing interval between when an input file is re-read (left), and when an output is re-used as the input for another job (right).

access frequencies, maintaining cache hit rates would require caching a fixed fraction of bytes stored. This design is not sustainable, since caches intentionally trade capacity for performance, and cache capacity grows slower than full data capacity. Fortunately, further analysis suggests more viable caching policies.

Figure 4 shows data access patterns plotted against file sizes. The distributions for fraction of jobs versus file size vary widely (top graphs), but converge in the upper right corner. In particular, 90% of jobs access files of less than a few GBs (note the log-scale axis). These files account for up to only 16% of bytes stored (bottom graphs). Thus, a viable cache policy would be to cache files whose size is less than a threshold. This policy would allow cache capacity growth rates to be detached from the growth rate in data.

Further analysis also suggest cache eviction policies. Figure 5 indicates the distribution of time intervals between data re-accesses. 75% of the re-accesses take place within 6 hours. Thus, a possible cache eviction policy would be to evict entire files that have not been accessed for longer than a workload specific threshold duration.

Figure 6 further shows that up to 78% of jobs involve data re-accesses (`CC-c`, `CC-d`, `CC-e`), while for other workloads, the fraction is lower. Thus, the same cache eviction policy potentially translates to different benefits for different workloads.

Future work should analyze file path semantics and hierarchy to see if the small data sets are stand-alone or samples of larger data sets. Frequent use of data samples would suggest opportunities to pre-generate data samples that preserve various kind of statistics. This analysis requires proprietary file name information, and would be possible within each MapReduce user's organization.

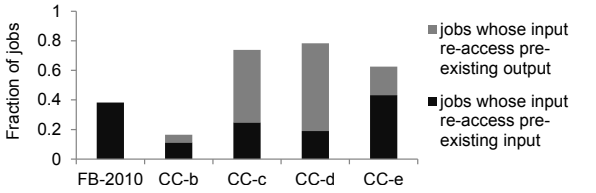

Figure 6: Fraction of jobs that reads pre-existing input path. Note that output path information is missing from `FB-2010`.

4 Workload Variation Over Time

The intensity of a MapReduce workload depends on the job submission rate, as well as the computation and data access patterns of the jobs that are submitted. System occupancy depends on the combination of these multiple time-varying dimensions. This section looks at workload variation over a week (§ 4.1), quantifies burstiness, a common feature for all workloads (§ 4.2), and computes temporal correlations between different workload dimensions (§ 4.3).

4.1 Weekly time series

Figure 7 depicts the time series of four dimensions of workload behavior over a week. The first three columns respectively represents the cumulative job counts, amount of I/O (again counted from MapReduce API), and computation time of the jobs submitted in that hour. The last column shows cluster utilization, which reflects how the cluster services the submitted workload describes by the preceding columns, and depends on the cluster hardware and execution environment.

The first feature to observe in the graphs of Figure 7 is that noise is high. This means that even though the number of jobs submitted is known, it is challenging to predict how many I/O and computation resources will be needed as a result. Also, standard signal process methods to quantify the signal to noise ratio would be challenging to apply to these time series, since neither the signal nor noise models are known.

Some workloads exhibit daily diurnal patterns, revealed by Fourier analysis, and for some cases, visually identifiable (e.g., jobs submission for `FB-2010`, utilization for `CC-e`). In Section 6, we combine this observation with several others to speculate that there is an emerging class of interactive and semi-streaming workloads.

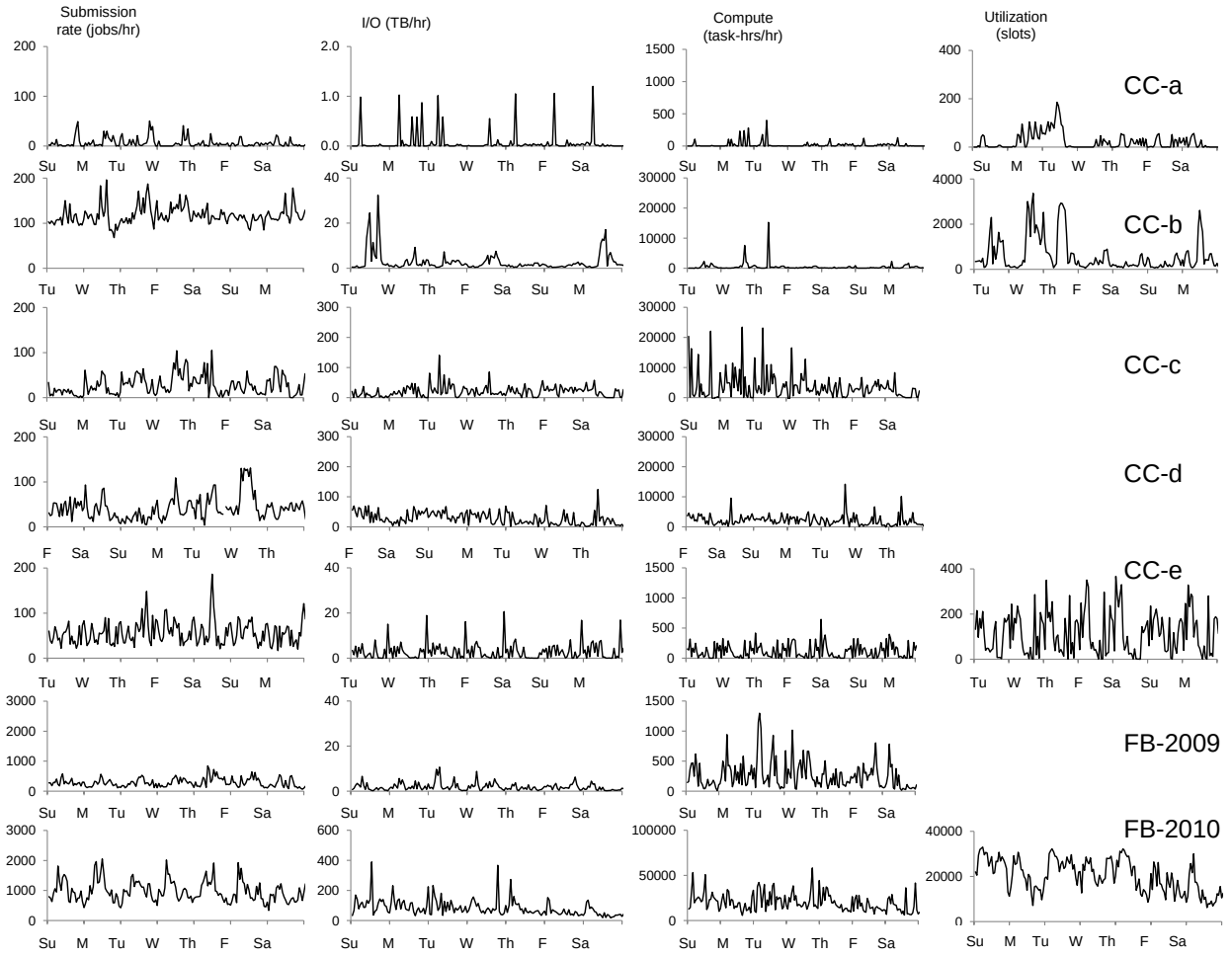

Figure 7: Workload behavior over a week. From left to right: (1) Jobs submitted per hour. (2) Aggregate I/O (i.e., input + shuffle + output) size of jobs submitted. (3) Aggregate map and reduce task time in task-hours of jobs submitted. (4) Cluster utilization in average active slots. From top row to bottom, showing `CC-a`, `CC-b`, `CC-c`, `CC-d`, `CC-e`, `FB-2009`, and `FB-2010` workloads. Note that for `CC-c`, `CC-d`, and `FB-2009`, the utilization data is not available from the traces. Also note that some time axes are misaligned due to short, week-long trace lengths (`CC-b` and `CC-e`), or gaps from missing data in the trace (`CC-d`).

4.2 Burstiness

Another feature of Figure 7 is the bursty submission patterns in all dimensions. Burstiness is an often discussed property of time-varying signals, but it is not precisely measured. A common metric is the peak-to-average ratio. There are also domain-specific metrics, such as for bursty packet loss on wireless links [38]. Here, we extend the concept of peak-to-average ratio to quantify a key workload property: burstiness.

We start defining burstiness first by using the median rather than the arithmetic mean as the measure of "average". Median is statistically robust against data outliers, i.e., extreme but rare bursts [26]. For two given workloads with the same median load, the one with higher peaks, that is, a higher peak-to-median ratio, is more bursty. We then observe that the peak-to-median ratio is the same as the 100^{th}-percentile-to-median ratio. While the median is statistically robust to outliers, the 100^{th}-percentile is not. This implies that the 99^{th}, 95^{th}, or 90^{th}-percentile should also be calculated. We extend this line of thought and compute the general n^{th}-percentile-to-median ratio for a workload. We can graph this vector of values, with $\frac{n^{th}-percentile}{median}$ on the x-axis, versus n on the y-axis. The resultant graph can be interpreted as a cumulative distribution of arrival rates per time unit, normalized by the median arrival rate. This graph is an indication of how bursty the time series is. A more horizontal line corresponds to a more bursty workload; a vertical line represents a workload with a constant arrival rate.

Figure 8 graphs this metric for one of the dimensions of our workloads. We also graph two different sinusoidal signals to illustrate how common signals appear under this burstiness metric. Figure 8 shows that for all work-

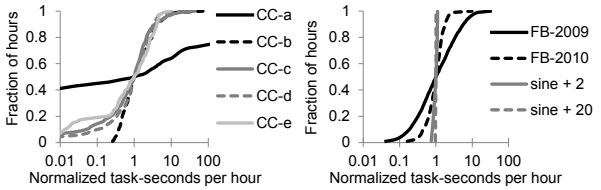

Figure 8: Workload burstiness. Showing cummulative distribution of task-time (sum of map time and reduce time) per hour. To allow comparison between workloads, all values have been normalized by the median task-time per hour for each workload. For comparison, we also show burstiness for artificial sine submit patterns, scaled with min-max range the same as mean (sine + 2) and 10% of mean (sine + 20).

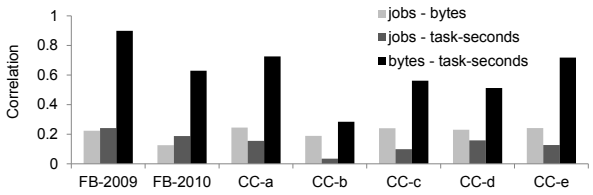

Figure 9: Correlation between different submission pattern time series. Showing pair-wise correlation between jobs per hour, (input + shuffle + output) bytes per hour, and (map + reduce) task times per hour.

loads, the highest and lowest submission rates are orders of magnitude from the median rate. This indicates a level of burstiness far above the workloads examined by prior work, which have more regular diurnal patterns [39, 30]. For the workloads here, scheduling and task placement policies will be essential under high load. Conversely, mechanisms for conserving energy would be beneficial during periods of low utilization.

For the Facebook workloads, over a year, the peak-to-median-ratio dropped from 31:1 to 9:1, accompanied by more internal organizations adopting MapReduce. This shows that multiplexing many workloads (workloads from many organizations) help decrease bustiness. However, the workload remains bursty.

4.3 Time series correlations

We also computed the correlation between the workload submission time series in all three dimensions, shown in Figure 9. The average temporal correlation between job submit and data size is 0.21; for job submit and compute time it is 0.14; for data size and compute time it is 0.62. The correlation between data size and compute time is by far the strongest. We can visually verify this by the 2^{nd} and 3^{rd} columns for `CC-e` in Figure 9. This indicates that MapReduce workloads remain data-centric rather than compute-centric. Also, schedulers and load balancers need to consider dimensions beyond number of active jobs.

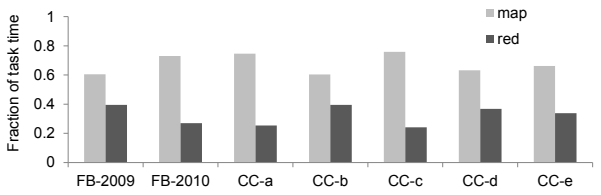

Figure 10: Aggregate fraction of all map and reduce task times for each workload.

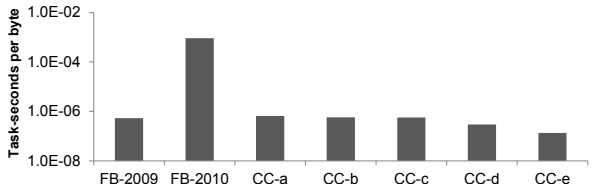

Figure 11: Task-seconds per byte for each workload.

5 Computation Patterns

MapReduce is designed as a combined storage and computation system. This section examines computation patterns by task times (§ 5.1), task granularity (§ 5.2), and common job types by both job names (§ 5.3.1) and a multi-dimensional analysis (§ 5.3.2).

5.1 Task times

Figure 10 shows the aggregate map and reduce task durations for each workload. On average, workloads spend 68% of time in map and 32% of time in reduce. For Facebook, the fraction of time spent mapping increases by 10% over a year. Thus, a design priority should be to optimize map task components, such as read locality, read bandwidth, and map output combiners.

Figure 11 shows for each workload the ratio of aggregate task durations (map time + reduce time) over the aggregate bytes (input + shuffle + output). This ratio aims to capture the amount of computation per data in the absence of CPU/disk/network level logs. The unit of measurement is task-seconds per byte. Task-seconds measures the computation time of multiple tasks, e.g., two tasks of 10 seconds equals 20 task-seconds. This is a good unit of measurement if parallelization overhead is small; it approximates the amount of computational resources required by a job that is agnostic to the degree of parallelism (e.g., X task-seconds divided into T_1 tasks equals to X task-seconds divided into T_2 tasks).

Figure 11 shows that the ratio of computation per data ranges from 1×10^{-7} to 7×10^{-7} task-seconds per byte, with the `FB-2010` workload having 9×10^{-4} task-seconds per byte. Task-seconds per byte clearly separates the workloads. A balanced system should be provisioned specifically to service the task-seconds per byte of a particular workload. Existing hardware benchmarks

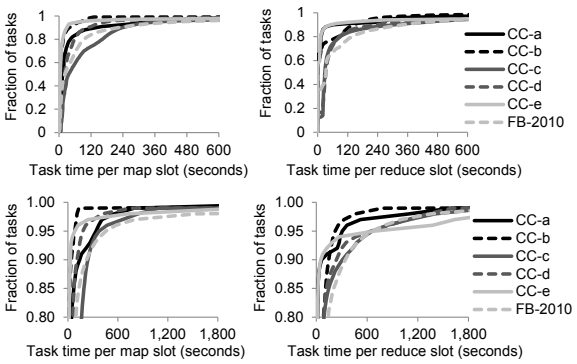

Figure 12: Task lengths per workload. Showing the entire distribution (top) and top 20% of the distribution (bottom).

should also probe a given value of this metric to be more relevant to MapReduce-like systems that combine data movements and computation.

5.2 Task granularity

Many MapReduce workload management mechanisms make decisions based on task-level information [41, 10, 11]. The underlying assumption is that different jobs break down into tasks in a regular fashion, with all tasks from all jobs being roughly "equal". The default policy in Hadoop MapReduce seeks to achieve this by assigning one map task per HDFS block of input, and one reduce task per 1GB of input. Many MapReduce operators override this policy, both intentionally and accidentally. Therefore, it is important to empirically verify the assumption of regular task granularity.

Figure 12 shows the cummulative distribution of task durations per workload. The distribution is long tailed. Approximately 50% of the tasks have durations of less than a minute. The remaining tasks have durations of up to hours. Thus, the traces do not support the assumption that tasks are regularly sized.

Absent an enforcement of task size, any task-level scheduling or placement decisions are likely to be suboptimal and prone to be intentionally undermined. For example, an operator with a very large job could divide her job into very large tasks. The fair scheduler ensures a fair share of task slots. During job submission troughs, this job would consume an increasingly large share of the cluster. When job submission peaks again, the large tasks would be incomplete. Absent preemptive task termination, the job with large tasks would have circumvented the intended fair share constraints. Furthermore, preemptive termination of long running tasks potentially results in large amounts of wasted work.

Task-level information does not form an inherent part of the workload, since it describes *how* MapReduce executes the jobs given, versus *what* the jobs actually are. MapReduce workload managers currently optimize execution scheduling and placement. They should also jobs decompose into regularly sized tasks.

5.3 Common job types

There are two complementary ways of grouping jobs: (1) by the job names submitted to MapReduce, which serves as a qualitative proxy for proprietary code, and (2) by the multi-dimensional job description according to per-job data sizes, duration, and task times, which serve as a quantitative proxy.

5.3.1 By job names

Job names are user-supplied strings recorded by MapReduce. Some computation frameworks built on top of MapReduce, such as Hive [1], Pig [3], and Oozie [2] generate the job names automatically. MapReduce does not currently impose any structure on job names. To simplify analysis, we focus on the first word of job names, ignoring any capitalization, numbers, or other symbols.

Figure 13 shows the most frequent first words in job names for each workload, weighted by number of jobs, the amount of I/O, and task-time. The `FB-2010` trace does not have this information. The top figure shows that the top handful of words account for a dominant majority of jobs. When these names are weighted by I/O, Hive queries such as `insert` and other data-centric jobs such as data extractors dominate; when weighted by task-time, the pattern is similar, unsurprising given the correlation between I/O and task-time.

Figure 13 also implies that each workload consists of only a small number of common computation types. The reason is that job names are either automatically generated, or assigned by human operators using informal but common conventions. Thus, job names beginning with the same word likely performs similar computation. The small number of computation types represent targets for static or even manual optimization. This would greatly simplify workload management problems, such as predicting job duration or resource use, and optimizing scheduling, placement, or task granularity.

Each workload services only a small number of MapReduce frameworks: Hive, Pig, Oozie, or similar layers on top of MapReduce. Figure 13 shows that for all workloads, two frameworks account for a dominant majority of jobs. There is ongoing research to achieve well-behaved multiplexing between different frameworks [27]. The data here suggests that multiplexing between two or three frameworks already covers the majority of jobs in all workloads here. We believe this observation to remain valid in the future. As new frameworks develop, enterprise MapReduce users are likely to

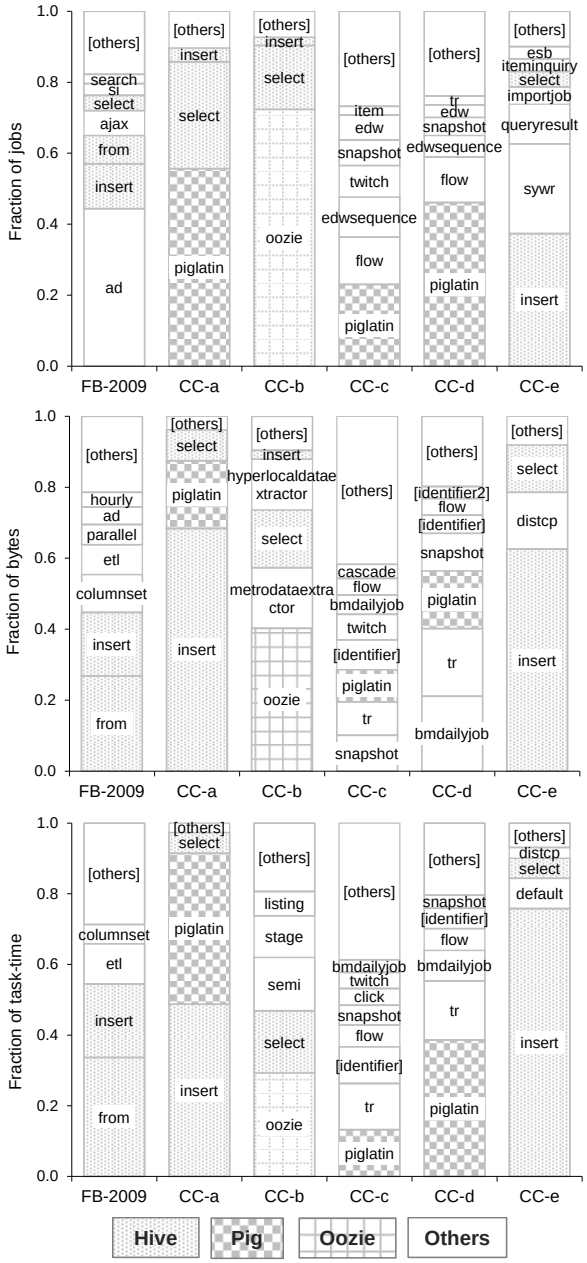

Figure 13: The first word of job names for each workload, weighted by the number of jobs beginning with each word (top), total I/O in bytes (middle), and map/reduce task-time (bottom). For example, 44% of jobs in the FB-2009 workload have a name beginning with "ad", a further 12% begin with "insert"; 27% of all I/O and 34% of total task-time comes from jobs with names that begin with "from" (middle and bottom). The FB-2010 trace did not contain job names.

converge on a revolving but small set of mature frameworks for business critical computations.

Future work should look beyond the first word of job names. Presently job name strings have no uniform structure, but contain pre- or postfixes such as dates, computation interactions, steps in multi-stage processing, etc., albeit with irregular format and ordering. Improved naming conventions, including UUIDs to identify multi-job workflows, would facilitate further analysis.

5.3.2 By multi-dimensional job behavior

Another way to group jobs is by their multi-dimensional behavior. Each job can be represented as a six-dimensional vector described by input size, shuffle size, output size, job duration, map task time, and reduce task time. One way to group similarly behaving jobs is to find clusters of vectors close to each other in the six-dimensional space. We use a standard data clustering algorithm, k-means [9]. K-means enables quick analysis of a large number of data points and facilitates intuitive labeling and interpretation of cluster centers [18, 33, 19].

We use a standard technique to choose k, the number of job type clusters for each workload: increment k until there is diminishing return in the decrease of intra-cluster variance, i.e., residual variance. Our previous work [18, 19] contains additional details of this methodology.

Table 4 summarizes our k-means analysis results. We have assigned labels using common terminology to describe the one or two data dimensions that separate job categories within a workload. A system optimizer would use the full numerical descriptions of cluster centroids.

We see that jobs touching $<$10GB of total data make up $>$92% of all jobs. These jobs are capable of achieving interactive latency for analysts, i.e., durations of less than a minute. The dominance of these jobs validate research efforts to improve the scheduling time and the interactive capability of large scale computation frameworks [28, 31, 14].

The dominance of small jobs complicates efforts to rein in stragglers [12], tasks that execute significantly slower than other tasks in a job and delay job completion. Comparing the job duration and task time columns indicate that small jobs contain only a handful of small tasks, sometimes a single map task and a single reduce task. Having few comparable tasks makes it difficult to detect stragglers, and also blurs the definition of a straggler. If the only task of a job runs slowly, it becomes impossible to tell whether the task is inherently slow, or abnormally slow. The importance of stragglers as a problem also requires re-assessment. Any stragglers would seriously hamper jobs that have a single wave of tasks. However, if it is the case that stragglers occur randomly with a fixed probability, fewer tasks per job means only a few jobs would be affected. We do not yet know whether stragglers occur randomly.

Interestingly, map functions in some jobs aggregates data, reduce functions in other jobs expands data, and many jobs contain data transformations in either stages. Such data ratios reverse the original intuition behind

	# Jobs	Input	Shuffle	Output	Duration	Map time	Reduce time	Label
CC-a	5525	51 MB	0	3.9 MB	39 sec	33	0	Small jobs
	194	14 GB	12 GB	10 GB	35 min	65,100	15,410	Transform
	31	1.2 TB	0	27 GB	2 hrs 30 min	437,615	0	Map only, huge
	9	273 GB	185 GB	21 MB	4 hrs 30 min	191,351	831,181	Transform and aggregate
CC-b	21210	4.6 KB	0	4.7 KB	23 sec	11	0	Small jobs
	1565	41 GB	10 GB	2.1 GB	4 min	15,837	12,392	Transform, small
	165	123 GB	43 GB	13 GB	6 min	36,265	31,389	Transform, medium
	31	4.7 TB	374 MB	24 MB	9 min	876,786	705	Aggregate and transform
	3	600 GB	1.6 GB	550 MB	6 hrs 45 min	3,092,977	230,976	Aggregate
CC-c	19975	5.7 GB	3.0 GB	200 MB	4 min	10,933	6,586	Small jobs
	477	1.0 TB	4.2 TB	920 GB	47 min	1,927,432	462,070	Transform, light reduce
	246	887 GB	57 GB	22 MB	4 hrs 14 min	569,391	158,930	Aggregate
	197	1.1 TB	3.7 TB	3.7 TB	53 min	1,895,403	886,347	Transform, heavy reduce
	105	32 GB	37 GB	2.4 GB	2 hrs 11 min	14,865,972	36,9846	Aggregate, large
	23	3.7 TB	562 GB	37 GB	17 hrs	9,779,062	14,989,871	Long jobs
	7	220 TB	18 GB	2.8 GB	5 hrs 15 min	66,839,710	758,957	Aggregate, huge
CC-d	12736	3.1 GB	753 MB	231 MB	67 sec	7,376	5,085	Small jobs
	214	633 GB	2.9 TB	332 GB	11 min	544,433	352,692	Expand and aggregate
	162	5.3 GB	6.1 TB	33 GB	23 min	2,011,911	910,673	Transform and aggregate
	128	1.0 TB	6.2 TB	6.7 TB	20 min	847,286	900,395	Expand and Transform
	43	17 GB	4.0 GB	1.7 GB	36 min	6,259,747	7,067	Aggregate
CC-e	10243	8.1 MB	0	970 KB	18 sec	15	0	Small jobs
	452	166 GB	180 GB	118 GB	31 min	35,606	38,194	Transform, large
	68	543 GB	502 GB	166 GB	2 hrs	115,077	108,745	Transform, very large
	20	3.0 TB	0	200 B	5 min	137,077	0	Map only summary
	7	6.7 TB	2.3 GB	6.7 TB	3 hrs 47 min	335,807	0	Map only transform
FB-2009	1081918	21 KB	0	871 KB	32 s	20	0	Small jobs
	37038	381 KB	0	1.9 GB	21 min	6,079	0	Load data, fast
	2070	10 KB	0	4.2 GB	1 hr 50 min	26,321	0	Load data, slow
	602	405 KB	0	447 GB	1 hr 10 min	66,657	0	Load data, large
	180	446 KB	0	1.1 TB	5 hrs 5 min	125,662	0	Load data, huge
	6035	230 GB	8.8 GB	491 MB	15 min	104,338	66,760	Aggregate, fast
	379	1.9 TB	502 MB	2.6 GB	30 min	348,942	76,736	Aggregate and expand
	159	418 GB	2.5 TB	45 GB	1 hr 25 min	1,076,089	974,395	Expand and aggregate
	793	255 GB	788 GB	1.6 GB	35 min	384,562	338,050	Data transform
	19	7.6 TB	51 GB	104 KB	55 min	4,843,452	853,911	Data summary
FB-2010	1145663	6.9 MB	600 B	60 KB	1 min	48	34	Small jobs
	7911	50 GB	0	61 GB	8 hrs	60,664	0	Map only transform, 8 hrs
	779	3.6 TB	0	4.4 TB	45 min	3,081,710	0	Map only transform, 45 min
	670	2.1 TB	0	2.7 GB	1 hr 20 min	9,457,592	0	Map only aggregate
	104	35 GB	0	3.5 GB	3 days	198,436	0	Map only transform, 3 days
	11491	1.5 TB	30 GB	2.2 GB	30 min	1,112,765	387,191	Aggregate
	1876	711 GB	2.6 TB	860 GB	2 hrs	1,618,792	2,056,439	Transform, 2 hrs
	454	9.0 TB	1.5 TB	1.2 TB	1 hr	1,795,682	818,344	Aggregate and transform
	169	2.7 TB	12 TB	260 GB	2 hrs 7 min	2,862,726	3,091,678	Expand and aggregate
	67	630 GB	1.2 TB	140 GB	18 hrs	1,545,220	18,144,174	Transform, 18 hrs

Table 4: Job types in each workload as identified by k-means clustering, with cluster sizes, centers, and labels. Map and reduce time are in task-seconds, i.e., a job with 2 map tasks of 10 seconds each has map time of 20 task-seconds. Note that the small jobs dominate all workloads.

map functions as expansions, i.e., "maps", and reduction functions as aggregates, i.e., "reduces" [23].

Also, map-only jobs appear in all but two workloads. They form 7% to 77% of all bytes, and 4% to 42% of all task times in their respective workloads. Some are Oozie launcher jobs and others are maintenance jobs that operate on very little data. Compared with other jobs, map-only jobs benefit less from datacenter networks optimized for shuffle patterns [7, 8, 25, 20].

The FB-2009 and FB-2010 workloads in Table 4 show that job types at Facebook changed significantly over one year. The small jobs remain, and several kinds of map-only jobs remain. However, the job profiles changed in several dimensions. Thus, for Facebook, any policy parameters need to be periodically revisited.

Future work should seek to perform k-means analysis for multiple workloads together, instead of for each workload separately. Such combined analysis would reveal what are the common job categories across workloads. This combined analysis requires normalizing each workload by its "size". Given that each workload contain different number of jobs, and have different ranges in each of the 6 dimensions, it is not yet clear how this "super k-means" analysis would proceed. A further innovation in analysis methodology awaits.

6 Data Analysis Trends

The non-trivial workloads we analyzed and our conversations with Facebook and Cloudera provide the opportunity to speculate on MapReduce-related data analysis trends, subject to verification as more comprehensive data becomes available. Overall, the trends reflect a desire for timely, high quality insights, extracted from

growing and complex data sets, and using as little technical expertise as possible.

Increasing semi-streaming analysis. Streaming analysis describes continuous computation processes, which often updates some time-aggregation metric [22]. For MapReduce, a common substitute for truly streaming analysis is to setup automated jobs that regularly operates on recent data. Since "recent" data is intentionally smaller than "historical" data, this type of semi-streaming analysis partially accounts for the large number of small jobs in all workloads.

Increasing interactive analysis. Another source of small jobs is interactive large-scale data analysis. Small jobs are capable of low latency, compared with larger jobs. Evidence suggesting interactive analysis include (1) Diurnal workload patterns, identified by both visual inspection and Fourier analysis, e.g., in jobs submitted per hour for `CC-c`, `FB-2009`, and `FB-2010`; (2) Presence across all workloads of frameworks such as Hive and Pig, one of whose design goals was ease of use by human analysts familiar with SQL. Not all small jobs in Hive and Pig would be interactively generated. However, over half of small jobs remain after we remove jobs whose names contain time stamps (semi-streaming analysis) and jobs submitted in the early morning (outside work hours).

At Facebook, a common use case is interactive data exploration using ad-hoc queries [17, 40]. Several recent research efforts at web search companies also focus on achieving interactive processing latency [31, 28].

Frameworks on top of MapReduce. All workloads include many jobs from Hive, Pig, or Oozie. Frameworks like these allow organizations to devote intellectual resources to understanding the data being processed, versus learning how to process the data using native map and reduce functions. Thus, these frameworks are likely to see increased adoption, both for established users such as Facebook, and especially for emerging users in media, retail, finance, manufacturing, and other industries.

Storage capacity as a constraint. The comparison between `FB-2009` and `FB-2010` workloads reveals orders of magnitude increase in per-job data sizes and the aggregate active data set touched. Such growth compels the heavy use of data compression and RAID-style error correction codes, which are more efficient with regard to storage capacity versus HDFS replication [15, 37]. As data generation and collection capabilities improve across industries, more MapReduce use cases would need to address the storage capacity challenge.

7 Summary and Conclusions

The analysis in this paper has several repercussions: (1) MapReduce has evolved to the point where performance claims should be qualified with the underlying workload assumptions, e.g., by replaying a suite of workloads. (2) System engineers should regularly re-assess designs priorities subject to changing use cases. Prerequisites to these efforts are workload replay tools and a public workload repository, so that engineers can share insights across different enterprise MapReduce deployments.

We have developed and deployed SWIM, a Statistical Workload Injector for MapReduce (`www.eecs.berkeley.edu/~ychen2/SWIM.html`). This is a set of workload replay tools, under New BSD License, that can pre-populate HDFS using synthetic data, scaled to cluster size, and replay the workload using synthetic MapReduce jobs. The workload replay methodology is further discussed in [18]. The SWIM repository already includes the `FB-2009` and `FB-2010` workloads. Cloudera has allowed us to contact the end customers directly and seek permission to make public their traces. We hope the repository can help others pursue the future work opportunities identified in the paper, and contribute to a scientific approach to designing large-scale data processing systems such as MapReduce.

7.1 Future work

There are many future opportunities for MapReduce workload characterization.

Analyze more workloads over longer time periods. Doing so allows us to improve the quality and generality of the derived design insights.

Analyze additional statistics. There always exists some additional analysis that can inform specific studies. In particular, the time between job submit and the first task being launched would reflect additional queuing bottlenecks; analysis of job submission patterns separated by submitters would distinguish human versus automated analysis; detailed accounting of task placement would reveal opportunities for improving data locality.

Automate analysis and monitoring tools. Enterprise MapReduce monitoring tools [21] should perform workload analysis automatically, present graphical results in a dashboard, ship only the anonymized/aggregated metrics workload comparison offsite, and improve tracing capability for Hive, Pig, HBase, and other such frameworks.

Create a MapReduce workload taxonomy. This would be possible once we have a large collection of insights from companies of different sizes and industries. Such a taxonomy would help identifying design priorities for future MapReduce-style systems. For the workloads analyzed here, the similarities we found could define them as a common class, while the differences could help distinguish sub-classes. Such a taxonomy requires more comprehensive data beyond what we analyzed in this paper.

We again invite the community to contribute additional workloads insights.

8 Acknowledgments

The authors are grateful for the feedback from our colleagues at UC Berkeley AMP Lab, Cloudera, and Facebook. We especially appreciate the inputs from Anthony Joseph, David Zats, Matei Zaharia, Jolly Chen, Todd Lipcon, Aaron T. Myers, John Wilkes, and Srikanth Kandula. This research is supported in part by AMP Lab (`https://amplab.cs.berkeley.edu/sponsors/`), and the DARPA- and SRC-funded MuSyC FCRP Multiscale Systems Center.

References

[1] Apache Hive. `http://hive.apache.org/`.

[2] Apache Oozie(TM) Workflow Scheduler for Hadoop. `http://incubator.apache.org/oozie/`.

[3] Apache Pig. `http://pig.apache.org/`.

[4] Gridmix. `HADOOP-HOME/mapred/src/benchmarks/gridmix` in Hadoop 0.21.0 onwards.

[5] Hadoop World 2011 Speakers. `http://www.hadoopworld.com/speakers/`.

[6] Sort benchmark home page. `http://sortbenchmark.org/`.

[7] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM 2008*.

[8] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *SIGCOMM 2010*.

[9] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, Cambridge, Massachusetts, 2004.

[10] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: coping with skewed content popularity in mapreduce clusters. In *Eurosys 2011*.

[11] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. PACMan: Coordinated Memory Caching for Parallel Jobs. In *NSDI 2012*.

[12] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the Outliers in Map-Reduce Clusters using Mantri. In *OSDI 2010*.

[13] L. N. Bairavasundaram, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, G. R. Goodson, and B. Schroeder. An analysis of data corruption in the storage stack. In *FAST 2008*.

[14] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer. Apache Hadoop goes realtime at Facebook. In *SIGMOD 2011*.

[15] D. Borthakur, R. Schmit, R. Vadali, S. Chen, and P. Kling. HDFS Raid. Tech talk. Yahoo Developer Network. 2010.

[16] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *INFOCOM 1999*.

[17] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy Efficiency for Large-Scale MapReduce Workloads with Significant Interactive Analysis. In *EuroSys 2012*.

[18] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In *MASCOTS 2011*.

[19] Y. Chen, K. Srinivasan, G. Goodson, and R. Katz. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *SOSP 2011*.

[20] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *SIGCOMM 2011*.

[21] Cloudera, Inc. Cloudera Manager Datasheet.

[22] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In *NSDI 2010*.

[23] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI 2004*.

[24] N. Feamster and H. Balakrishnan. Detecting bgp configuration faults with static analysis. In *NSDI'05*.

[25] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *SIGCOMM 2009*.

[26] J. Hellerstein. Quantitative data cleaning for large databases. Technical report, United Nations Economic Commission for Europe, February 2008.

[27] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI 2011*.

[28] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In *SOSP 2009*.

[29] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic. In *SIGCOMM 1993*.

[30] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *ISCA 2011*.

[31] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of web-scale datasets. In *VLDB 2010*.

[32] M. Mesnier, E. Thereska, G. R. Ganger, and D. Ellard. File classification in self-* storage systems. In *ICAC 2004*.

[33] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das. Towards characterizing cloud backend workloads: insights from google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37:34–41, March 2010.

[34] J. C. Mogul. The case for persistent-connection http. In *SIGCOMM 1995*.

[35] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson. A trace-driven analysis of the unix 4.2 bsd file system. In *SOSP 1985*.

[36] V. Paxson. End-to-end internet packet dynamics. In *SIGCOMM 1997*.

[37] A. Ryan. Next-Generation Hadoop Operations. Bay Area Hadoop User Group, February 2010.

[38] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis. The β-factor: measuring wireless link burstiness. In *SenSys 2008*.

[39] E. Thereska, A. Donnelly, and D. Narayanan. Sierra: practical power-proportionality for data center storage. In *EuroSys 2011*.

[40] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at Facebook. In *SIGMOD 2010*.

[41] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys 2010*.