An Analysis of Traces from a Production MapReduce Cluster

Soila Kavulya, Jiaqi Tan, Rajeev Gandhi and Priya Narasimhan

Parallel Data Laboratory

Carnegie Mellon University

Pittsburgh, PA 15213-3890

Abstract

MapReduce is a programming paradigm for parallel processing that is increasingly being used for data-intensive applications in cloud computing environments. An understanding of the characteristics of workloads running in MapReduce environments benefits both the service providers in the cloud and users: the service provider can use this knowledge to make better scheduling decisions, while the user can learn what aspects of their jobs impact performance. This paper analyzes 10-months of MapReduce logs from the M45 supercomputing cluster which Yahoo! made freely available to select universities for systems research. We characterized resource utilization patterns, job patterns, and sources of failures. We use an instance-based learning technique that exploits temporal locality to predict job completion times from historical data and identify potential performance problems in our dataset.

1 Introduction

Large-scale data processing (cloud computing) is becoming increasingly common, and has been facilitated by frameworks such as Google's MapReduce [1], which parallelizes and distributes jobs across large clusters. In particular, Hadoop, the open-source implementation of MapReduce, has been widely used for large-scale data-intensive tasks such as click-log mining, web crawling, image processing, and data analysis. Hadoop is widely used at companies such as Yahoo!, Facebook, and Fox Interactive Media, as well as for academic research [2]. MapReduce clusters process large amounts of data–at Google alone, more than 100,000 MapReduce jobs process more than 20 PB of data daily [1].

Frameworks such as Hadoop allow users to harness dedicated or virtualized resources in computing clouds to run their data-intensive jobs. The pay-per-use cost model of cloud computing (e.g., commercial datacenters like Amazon's Elastic Compute Cloud (EC2) charge $0.10-0.80/hour/node for users wishing to use/lease the computation and storage resources) coupled with the scale of the clusters required or instantiated by the users makes cost-management essential. An understanding of the characteristics of workloads running in MapReduce environments as well as the factors affecting job-completion times can benefit both the cloud-computing service provider as well as the users: the service provider can use this knowledge to make better scheduling decisions and to provision resources more effectively across diverse workloads, while the user can learn which aspects of their jobs impact their performance and drive the cost of leasing the cloud-computing resources.

To gain insight on MapReduce workloads, we analyzed 10-months of trace data from the M45 [3] supercomputing cluster, a production Hadoop environment that Yahoo! administers and has made freely available to select universities for systems research. The M45 cluster has approximately 400 nodes, 4000 processors, 3 terabytes of memory, and 1.5 petabytes of disk space. The cluster runs Hadoop, and uses Hadoop on Demand (HOD) to provision virtual Hadoop clusters over the large physical cluster. For the past year, researchers at Carnegie Mellon University have been running diverse data-intensive workloads on M45, such as large-scale graph mining, text and web mining, large-scale computer graphics, natural language processing, machine translation problems, and data-intensive file system applications. This paper describes our analysis of the M45 trace data–both black-box OS performance data (such as CPU utilization) and white-box performance data (extracted from native Hadoop logs)–obtained over a 10-month period spanning parts of 2008 and 2009.

The primary contribution of this paper is to provide a description of the statistical properties of this trace data that will aid other researchers in understanding the performance and failure characteristics of Hadoop jobs running in large-scale real-world clusters for different, often unknown, workloads. The main insights from our analysis, as shown in Table 1, are that: (i) the job completion times and cluster allocation patterns follow a long-tailed distribution; (ii) better diagnosis and recovery approaches are needed to reduce error latencies in long-running tasks; (iii) the evenly-balanced load across most jobs implies that peer-comparison might be a suitable strategy for anomaly detection; and (iv) the low variability in user behavior over short periods of time allows us to exploit temporal locality to predict job completion times.

A secondary contribution of this paper is to understand which aspects of a Hadoop job most affect the completion time of that job, and to present a simple analytical model with configurable Hadoop-specific parameters to predict job-completion times. Due to the lack of labeled data, we could not verify performance problems. Instead, we inferred performance problems by predicting job completion times and flagging large prediction errors as potential performance problems or workload changes.

The paper is organized as follows: Section 2 provides a brief background on Hadoop while Section 3 and 3.2 describe the M45 data traces and the cluster allocation patterns. Section 4 provides an analysis of the traces, including job inter-arrival times and completion times, and job structure. Section 6 presents our prediction algorithm. Section 7 evaluates our approach. Section 8 compares our results to related work. Section 9 concludes.

Table 1: Summary of Findings

Finding	Implications
Workload characterization	
I. The average resource utilization on the cluster was low. For example, average CPU utilization across all nodes ranged from 5% to 10% *(Section 3.2)*.	The low resource utilization presents an opportunity to exploit energy-aware job scheduling to reduce power consumption.
II. The job completion times and cluster allocation patterns follow a long-tailed distribution. *(Figures 2(c) and 3(b))*	Fair job schedulers, *e.g.*, the Hadoop capacity scheduler, can prevent large jobs or heavy users from monopolizing the cluster.
III. Most failed jobs abort within 150 seconds from the first aborted task. However, we observed a maximum error latency of 4.3 days *(Section 4.1)*.	Better diagnosis and recovery approaches are needed to reduce error latencies in long-running tasks.
IV. Low Gini Coefficients for Map and Reduce task durations in most jobs indicates that work was generally evenly distributed across tasks *(Section 5.3.1)*.	Peer-comparison might be a feasible strategy for anomaly-detection *(Section 7.2)*.
Performance prediction	
I. The variability in user behavior over short periods of time was low as users tend to run the same job repeatedly over short intervals of time *(Section 6)*.	Exploiting temporal locality can improve the accuracy of performance prediction.
II. Locally-weighted linear regression predicts job completion times better than the distance-weighted algorithm when data input sizes are scaled. *(Section 7)*.	Large prediction errors can be used to detect performance problems and workload changes *(Section 7.1)*.

2 Background

Hadoop [4] is an open-source implementation of Google's MapReduce [5] framework that enables distributed, data-intensive, parallel applications by decomposing a massive job into smaller (Map and Reduce) tasks and a massive data-set into smaller partitions, such that each task processes a different partition in parallel.

A Hadoop job consists of a group of Map and Reduce tasks performing some data-intensive computation. The Map task executes a user-defined map function for each key/value pair in its input. The Reduce task consists of a shuffle, sort, and reduce phase. During the shuffle and sort phase, the Reduce task fetches, merges, and sorts the outputs from completed map tasks. Once all the data is fetched and sorted, the Reduce task calls a user-defined function for each input key and list of corresponding values.

Hadoop shares data amongst the distributed tasks in the system through the Hadoop Distributed File System (HDFS), an implementation of the Google Filesystem [6]. HDFS splits and stores files as fixed-size blocks (except for the last block).

Hadoop uses a master-slave architecture with a unique master node and multiple slave nodes, as shown in Figure 1. The master node typically runs two daemons: (1) the JobTracker that schedules and manages all of the tasks belonging to a running job; and (2) the NameNode that manages the HDFS namespace by providing a filename-to-block mapping, and regulates access to files by clients (i.e., the executing tasks). Each slave node runs two daemons: (1) the TaskTracker that launches tasks on its local node, and tracks the progress of each task on its node; and (2) the DataNode that serves data blocks (on its local disk) to HDFS clients.
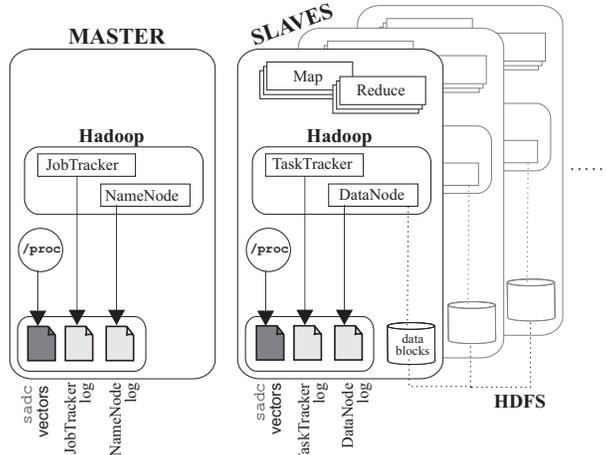
Figure 1: Architecture of Hadoop, showing the instrumentation sources for our M45 data traces.

3 Description of dataset

The data we collected spanned an 10-month period from April 25, 2008 to April 24, 2009. (*The Hadoop logs from Nov 12, 2008 to Jan 18, 2009 were unavailable.*) The dataset comprised of 171,079 Hadoop jobs run by 31 different users. The users belonged to the following research groups:

1. *Natural Language Processing (NLP):* This group ran machine translation and other natural language processing tasks, and accounted for 47% of active users.

2. *Data Mining:* This group ran large-scale graph mining, large-scale computer graphics, and text and web mining applications, and accounted for 31% of active users.

3. *Systems Research:* This group ran file-system benchmarks, and log-analysis tools, and accounted for 22% of active users.

Table 2 gives an overview of the dataset. User jobs completed in 20 minutes on average–with less than 4% of jobs exceeding 30 minutes. The maximum job completion time that we observed was 6.83 days. Each job consisted of an average of 153 Maps and 19 Reduces running on 27 nodes. The maximum number of nodes allocated to a job in our dataset was 299. We categorized the completion status of jobs as: (i) successful jobs, (ii) failed jobs which were aborted by the JobTracker due to unhandled exceptions, and (iii) cancelled jobs which were aborted by the user or the weekly maintenance daemon on M45. The job success-rate was high–97% of jobs completed successfully, 2.4% failed, and 0.6% were cancelled.

The cluster ran three major Hadoop versions during this period (see Table 2) namely: (i) Hadoop version 0.16 from April 25, 2008, (ii) Hadoop version 0.17 from June 25, 2008, and (iii) Hadoop version 0.18 from January 21, 2009. There were also some minor version upgrades during this period.

3.1 Data collection

Figure 1 shows the native Hadoop logs (labeled the JobTracker logs) that we obtained over the 10-month period for our analysis. We parsed the JobTracker logs to extract the information listed in Table 3. In addition to the log traces, we also obtained and analyzed periodically sampled `/proc`-based OS performance data (labeled black-box data) over the same period of time. The OS performance data was sampled at 5-minute intervals and a subset of the metrics collected is listed in Table 4.

3

Table 2: Summary of M45 dataset.

Log Period	Apr 25 - Nov 12, 2008
	Jan 19 - Apr 24, 2009
Hadoop versions	0.16: Apr 25, 2008 -
	0.17: Jun 25, 2008 -
	0.18: Jan 21, 2009 -
Number of active users	31
Number of jobs	171079
Successful jobs	165948 (97%)
Failed jobs	4100 (2.4%)
Cancelled jobs	1031 (0.6%)
Average maps per job	$154 \pm 558\sigma$
Average reduces per job	$19 \pm 145\sigma$
Average nodes per job	$27 \pm 22\sigma$
Maximum nodes per job	299
Average job duration	$1214\pm13875\sigma$ seconds
Maximum job duration	6.84 days
Node days used	132624

Table 3: Job history statistics.

Metric	Description
Task duration	Duration of Maps/Reduces
Task status	Success, failed, or incomplete
Task input/output	Input/output records (or bytes)
Spilled records	Records spilled to disk
Data locality	Data, rack, or non-local
Combiner records	Combiner input/output records
File-system bytes	Bytes read/written to disk (or HDFS)
Map/Reduce counts	Launched tasks and Map/Reduce slots

Table 4: Resource usage metrics

Resource	Metrics
CPU utilization	System, User, IOWait
Network	KBps received/sent
Disk	KBps read/written

## 3.2 Resource Utilization

Resource utilization on the cluster rose significantly from 2008 to 2009. The allocation of the cluster increased from ~20% of the node hours available in mid-2008 to ~40% in late 2008 and 2009 (see Figure 2(a)). CPU utilization on the cluster increased from ~5% in 2008 to ~10% in 2009. This increase in CPU utilization does not appear to be correlated with the increased node allocation in November 2008. Since the increase in CPU utilization corresponds with an upgrade from Hadoop 0.17 to Hadoop 0.18, we hypothesize that the increased CPU utilization might be due to changes in the compression of map outputs in Hadoop 0.18, coupled with changes in user workload. Network and disk utilization also rose from 2008 to 2009 (see Table 5). Despite the increased resource utilization in 2009, the cluster operated below its peak capacity. The low resource utilization presents an opportunity to reduce power consumption by exploiting energy-aware job scheduling.

Node allocation patterns between users was skewed with 32% of the users accounting for 98% of the node hours allocated in the cluster (see Figure 2(c)).

Table 5: Network and Disk Utilization

Metric	2008 ($\mu \pm \sigma$)	2009 ($\mu \pm \sigma$)
Network KBps Received	150±213	494±402
Network KBps Sent	148±217	501±405
Disk KBps Read	308±180	960±890
Disk KBps Write	361±369	987±1300
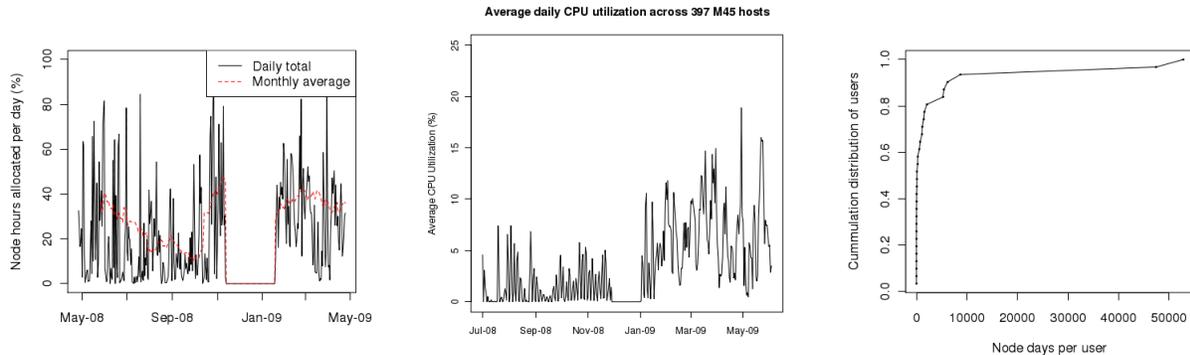
(a) Mean node hours allocated increased from ~20% in mid-2008 to ~40% in late 2008.

(b) Average CPU utilization increased to ~10% in 2009. This increase appears correlated with upgrade to Hadoop 0.18.

(c) Cluster allocation patterns follow a long-tailed distribution– 32% of the users account for 98% of the node allocations.

Figure 2: Resource usage patterns on M45.

Table 6: Distribution of job completion time (M - Mean, SD - Standard deviation, CV - Coefficient of Variation, KS - maximal distance between the cumulative distribution function of the theoretical distribution and the sample empirical distribution).

Period	Job status	Mean	SD	CV	Best fitted distribution	KS
2008	Successful	1065.35	11673.79	10.96	Lognormal (μ=4.29, σ=1.44)	0.130
	Failed	7193.96	38521.66	5.36	Lognormal (μ=4.23, σ=2.22)	0.300
	Cancelled	45861.79	123114.3	2.68	Lognormal (μ=6.75, σ=2.84)	0.128
2009	Successful	507.375	1268.315	2.50	Lognormal (μ=5.24, σ=1.44)	0.060
	Failed	1273.24	5574.86	4.38	Lognormal (μ=4.87, σ=1.88)	0.200
	Cancelled	7335.54	37860.92	5.16	Lognormal (μ=6.78, σ=1.87)	0.057

4   Job Characterization

We analyzed the completion times of successful jobs, failed jobs, and cancelled jobs in 2008 and 2009. Failed jobs are jobs which were aborted by the JobTracker due to unhandled exceptions, and cancelled jobs were aborted by the user or the weekly maintenance daemon. We measured the completion time of cancelled jobs by subtracting the timestamp of the last entry in the JobTracker log file from the job submisstion time. Figure 3(a) shows the total number of successful, failed and cancelled jobs per month. There was a burn-in period over the first two months of operation during which the job failure rate was high–in April 2008, 70% of jobs failed, and in May 2008, 10% of jobs failed. From June 2008 onwards, the failure rate dropped to 3%.

We computed the mean, standard deviation, and the coefficient of variation (CV) for the job completion times. The coefficient of variation is the ratio of the standard deviation to the mean. Distributions with CV < 1 have a low-variance, while those with CV > 1 have high-variance. We used the maximum-likelihood estimation method in R's MASS[7] package to fit the job completion times to the exponential, Weibull, and lognormal distributions. We measured the goodness of fit between the empirical distribution function and the reference distribution using the Kolmogorov-Smirnov test.

We observed that the job completion times follow a long-tailed distribution with 95% of jobs completing within 20 minutes. (see Figures 3(b)). The longest running job we observed lasted 6.83 days. Jobs aborted by the JobTracker had lower completion times than jobs aborted by the user. 90% of failed jobs were
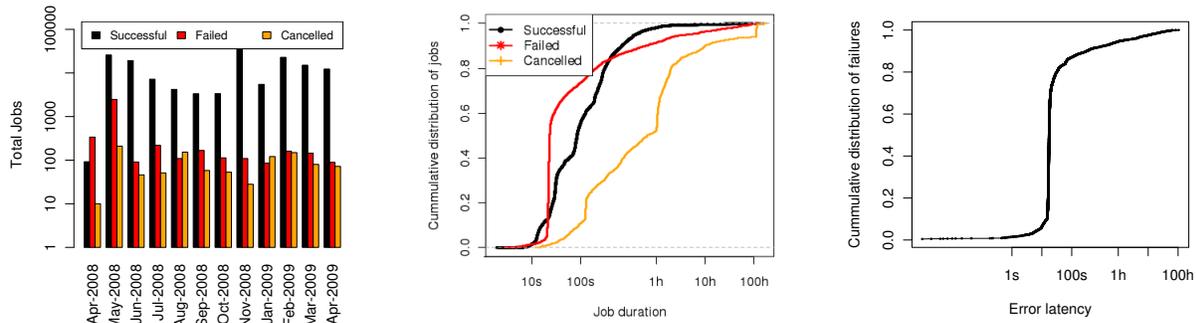
(a) The job failure rate during the first two months was high, but subsequently dropped to 3%.

(b) Job completion times follow a long-tailed distribution.

(c) Most jobs fail within 150 seconds after the first unrecoverable task aborts. However, we observed a maximum error latency of 4.3 days.

Figure 3: Distribution of job completion times and error latencies.

aborted within 35 minutes, compared to cancelled jobs where 90% of jobs were aborted within 9 hours. In Figure 3(b), we observe a cluster of cancelled jobs which were aborted after 100hrs. These completion times correspond to the weekly maintenance task on M45 which shuts down active Hadoop daemons.

We measured the goodness of fit of the job completion times against the exponential, Weibull, and lognormal distributions. The lognormal distribution was the best fitting distribution for the successful, failed and cancelled jobs (see Table 6). However, the large distances yielded by the Kolmogorov-Smirnoff test for failed jobs, indicates that a distribution other than the lognormal, Weibull, and exponential distributions that we tested might be a better fit.

The long-tailed job completion times and cluster allocation patterns in our dataset motivate the need for fair job schedulers, e.g., the Hadoop capacity scheduler [8] to prevent large jobs or heavy users from monopolizing the cluster.

4.1 Failure Characterization

There were 4100 failed jobs and 1031 incomplete jobs in our dataset. These failures accounted for 3% of the total jobs run. We observed that:

1. *Most jobs fail within 150 seconds after the first aborted task.* Figure 3(c) shows that 90% of jobs exhibited an error latency of less than 150 seconds from the first aborted task to the last retry of that task (the default number of retries for aborted tasks was 4). We observed a maximum error latency of 4.3 days due to a copy failure in a single reduce task. Better diagnosis and recovery approaches are needed to reduce error latencies in long-running tasks.

2. *Most failures occurred during the map phase.* Failures due to task exceptions in the map phase were the most prevalent–36% of these failures were due to array indexing errors (see Figure 4). IO exceptions were common in the reduce phase accounting for 23% of failures. Configuration problems, such as missing files, led to failures during job initialization.

3. *Application hangs and node failures were more prevalent in cancelled jobs.* Task timeouts, which occur when a task hangs and fails to report progress for more than 10 minutes, and lost TaskTracker daemons due to node or process failures were more prevalent in cancelled jobs than in failed jobs.
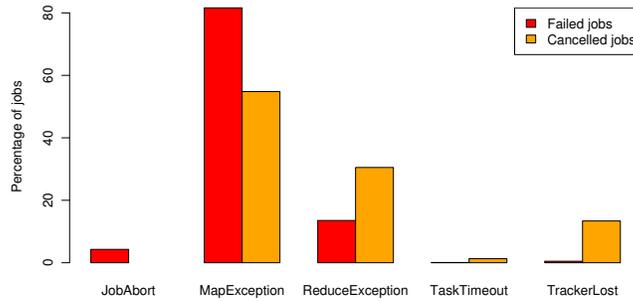
6

Figure 4: Most job failures occurred during the map phase.

4. *Spurious exceptions exist in the logs.* The logs contained spurious exceptions due to debugging statements that were turned on by default in certain versions of Hadoop. For example, a debug exception to troubleshoot a problem in the DFS client attributed to data buffering [9], and an exception due to a disabled feature that ignored the non-zero exit codes in Hadoop streaming, accounted for 90% of exceptions from January 21 to February 18, 2009. This motivates the need for error-log analysis tools that highlight important exceptions to users.

5 Job Structure

We categorized jobs based on the amount of time spent in each MapReduce phase namely: (i) map-only jobs, (ii) map-mostly jobs, (iii) shuffle-mostly jobs, and (iv) reduce-mostly jobs.

1. *Map-only jobs:* Map-only jobs had no reduce phase and were the most common pattern in our dataset with 77% of jobs falling in this category. This pattern was typically used by the natural language processing group.

2. *Map-mostly jobs:* These jobs had a reduce phase but spent most of their time in the map-phase. This pattern accounted for 14% of jobs and were run by the data mining and natural language processing groups.

3. *Shuffle-mostly jobs:* These jobs spent most of their time in the shuffle phase and accounted for 2% of the jobs. These jobs were typically run by the data mining group.

4. *Reduce-mostly jobs:* These jobs spent most of their time in the reduce phase and accounted for 7% of the jobs. The pattern was used for parameter estimation in data mining, building vocabularies for natural language processing, and file-system evaluation.

We also studied the characteristics of the behavior of individual MapReduce jobs by utilizing our prior work in Hadoop log analysis [10, 11] to generate abstractions and visualizations of MapReduce behavior within individual jobs at the level of Maps and Reduces, particularly making use of the "Swimlanes" space-time plots of task execution in a Hadoop cluster.
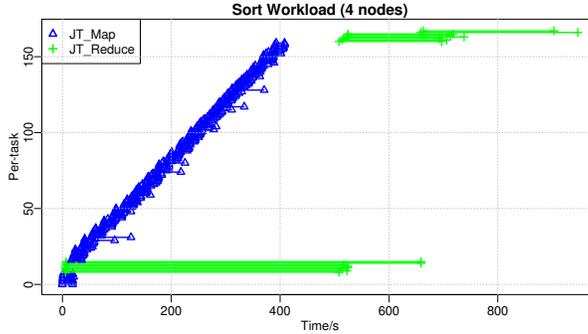
Figure 5: Example Swimlanes plot showing two jobs executing one after another. Maps are shown in blue with crosses at the end of each line, and Reduces are shown in green with triangles at the end of each line. The horizontal axis shows time elapsed during the job, and each horizontal line shows the duration of execution of one task (Map or Reduce). Each tick in the vertical axis is occupied by one task.

## 5.1 In-Job Behavior: "Swimlanes" Plots

The Swimlanes visualization plots the executions of Map and Reduce tasks of one (or more) MapReduce job(s). Figure 5 shows a plot of a typical MapReduce job: the horizontal axis shows the time elapsed since the start of the job. For each (Map or Reduce) task, a horizontal line is plotted to indicate the duration of the job during which it was executing. The executions of Map and Reduce tasks are plotted using different colors. Tasks are plotted in the order in which they were executed during the job. ([11] explores other insights that can be gleaned from different orderings of the tasks being plotted.) The behavior of each MapReduce job can be studied in greater detail from the Swimlanes plot of its execution, and we focus on two aspects of MapReduce behavior that are highlighted by our visualization: (i) the phenomenon of the execution of tasks progressing in "waves", and (ii) the equality of the distribution of job times.

5.2 "Wave" Behavior of Task Scheduling

Hadoop MapReduce slave nodes are configurable to concurrently execute up to a particular number of Map (and Reduce) tasks[1]. Hence, slave nodes are typically said to possess that number of Map (and Reduce) slots, each of which can execute one Map (and Reduce) task. If the number of Map (or Reduce) tasks in the job exceeds the number of Map (or Reduce) slots available, then Maps (or Reduces) are first scheduled to execute on all available slots are first and these Maps (or Reduces) form the first "wave" of tasks, and subsequent tasks form the second, third, and subsequent waves. We labeled tasks by the waves they belonged to, and illustrated these waves on our Swimlanes plots as well. Such "wave" behavior can be seen in Figure 5 where there are multiple waves of Map tasks, and two waves of Reduce tasks.

Understanding the "wave" behavior in tasks, such as the number of waves, and the sizes of waves, would aid the configuration of tasks for improved cluster utilization. For instance, having waves that do not saturate all available slots in the cluster would result in a portion of the cluster being idle during that wave (in particular, this occurs frequently during the last wave of a job).

---

[1]As of Hadoop release 0.20, Map and Reduce task slots can be used interchangeably and there is no longer a notion of dedicated Map-slots and Reduce-slots.
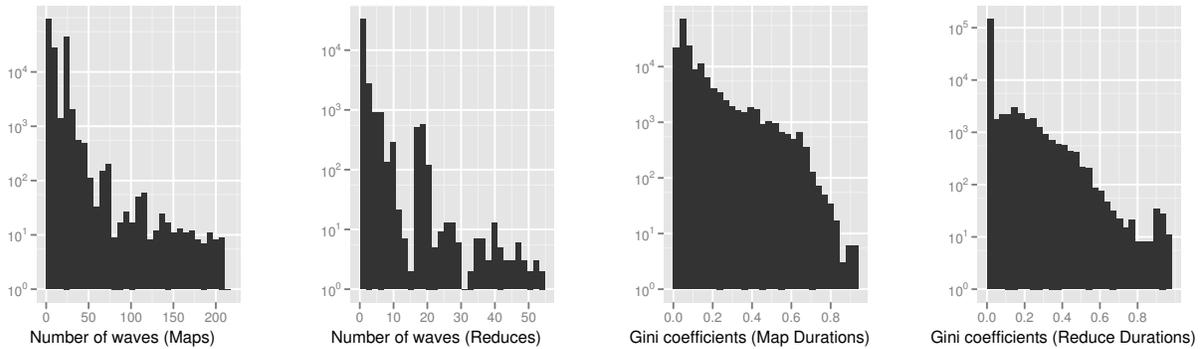
(a) Histogram of Number of Map Waves in Jobs.

(b) Histogram of Number of Reduce Waves in Jobs.

(c) Histogram of Gini Coefficients of Map Durations.

(d) Histogram of Gini Coefficients of Reduce Durations.

Figure 6: Histograms of Map waves, Reduce waves, and Gini Coefficients for jobs (Frequencies shown on a logarithmic scale).

5.2.1 Observations of Wave Behavior

First, we survey the distribution of the number of Map waves and Reduce waves in user jobs in our dataset. Figure 6 shows the distribution of jobs and the number of Map waves and Reduce waves in jobs in our dataset. We observed that most jobs have relatively few Map waves (more than 95% of jobs have fewer than 24 waves of Maps), while a small number of jobs have a large number of Map waves (several hundred). On the other hand, 95% of jobs have fewer than 7 waves of Reduces, with the vast majority of jobs having fewer than 2 Reduce waves. We hypothesize that jobs with few (< 24) Map waves exhibit these waves due to users provisioning virtual Hadoop clusters that are small relative to the number of available nodes out of politeness; this resulted in their jobs requiring multiple waves to complete the execution of all Map tasks. Jobs with extremely large numbers of Map waves (in excess of 100) are likely to have been processing very large datasets that resulted in a large number of input splits, resulting in large numbers of Map tasks. The observation that almost all jobs have 2 or fewer Reduce waves probably reflects users using a published heuristic for setting the number of Reduce tasks in a job to 0.95 the number of available Reduce slots [12]. Hence, most Map tasks occur in multiple waves, while Reduce tasks tend to complete in one to two waves, in most real-world workloads.

5.3 Equity of Task Durations

Next, we study the extent to which tasks (Maps and Reduces) in a MapReduce job take comparable amounts of time to complete. This property of how equitable durations are across tasks, which we term the "equity of task durations", is important in optimizing the performance of parallel programs in general. Intuitively, parallel programs perform optimally when the different threads (a term we use loosely to refer to independently executing parts) of the program executing in parallel complete in the same amount of time; otherwise, the runtime of the program can be reduced by redistributing work from threads taking longer to complete to work taking less time to complete. Alternatively, the disequity of task durations, or imbalances in runtimes of the various tasks, can be seen as indications of inefficiency or performance problems (if each task has the same amount of work), or as an indication that the job at hand intrinsically consists of barriers to its parallelization. Visually, this can also be seen from the Swimlanes plots of MapReduce job behavior; "straggler" Map or Reduce tasks, which take much more time than other tasks to complete, slow the completion of jobs, and are easily seen visually from our plots. These stragglers would also typically render the durations of

9

tasks in the job less equitable, so studying the equity of task durations is also instructive in indirectly (but scalably, for large datasets) identifying the possible existence of straggler tasks.

5.3.1   Measure of Task Equity: Gini Coefficient

We use the Gini Coefficient [13], a measure of statistical dispersion typically used as an indicator for income inequality in nations, as a measure of the extent to which task durations are comparable (or equal in the best case). The Gini Coefficient runs from 0.0 to 1.0 in value and is a ratio of actual cumulative job durations to the ideal (equal) cumulative job durations. In our context, a Gini Coefficient value of 0.0 indicates that all tasks have equal durations, while higher values indicate that there is a greater disparity among the durations of tasks. The Gini Coefficient is more useful than observing the Cumulative Distribution Function (CDF) of task runtimes when analyzing large datasets of job data, such as with our dataset, as it provides us with a single number indicative of the "health" of the equity of task durations for each job. However, the Gini Coefficient only serves as a measure of (statistical) dispersion, but does not imply if a particular level of dispersion is desirable (or not).

5.3.2   Observations of Equity of Task Durations

Next, we surveyed the Gini Coefficients of the durations of Map and Reduce tasks for each job in our dataset. Figure 6 shows the histograms of the number of jobs with the given Gini Coefficient values for their Map and Reduce tasks separately.

We observed that most jobs exhibited low values of the Gini Coefficient for both Map and Reduce task durations, indicating that the durations of Map tasks (and Reduce tasks) were comparable in most jobs, with more than 85% of jobs with Gini Coefficients of Map durations of < 0.2, and with more than 75% of jobs with Gini Coefficients of Reduce durations of < 0.2. However, we also observed that there were jobs with high values of Gini Coefficients in excess of 0.5, could be indicative of performance problems, or data skews. Hence, we can conclude that most user jobs in M45 had Maps and Reduces with runtimes that were comparable amongst each other.

6   Performance Prediction

Anecdotal evidence from users indicates that they experienced performance problems while running their jobs. Due to the lack of labeled data on performance problems, we inferred performance problems by predicting job completion times and flagging large prediction errors as potential performance problems or workload changes.

The performance prediction algorithm extends the instance-based (nearest-neighbor) learning approaches described in [14, 15]. The algorithm consists of two steps: (i) finding a set of similar jobs in the recent past, and (ii) generating regression models which predict the completion time of the incoming job from the Hadoop-specific input parameters listed in the set of similar jobs. A manual inspection of jobs corroborated the findings of [16] which showed that users tended to run the same job repeatedly over short intervals of time–allowing us to exploit temporal locality to predict job completion times.

6.1   Finding similar jobs

We used the Hadoop-specific input parameters listed in Table 7 to identify similar jobs in the recent past. These features are known at the onset of the incoming job and are used to quantify the distance between jobs. We included the job submission time as a feature in order to exploit temporal locality by apportioning smaller distances to recent jobs.

Table 7: Distance Computation and Regression Features.

Feature	Type	Distance	Regression
Job submission time	Numeric	$\checkmark$	$\times$
User name	Categorical	$\checkmark$	$\times$
Job name	Categorical	$\checkmark$	$\times$
Map slots	Numeric	$\checkmark$	$\checkmark$
Reduce slots	Numeric	$\checkmark$	$\checkmark$
Map input bytes	Numeric	$\checkmark$	$\checkmark$
Map input records	Numeric	$\checkmark$	$\checkmark$

We used the Heterogeneous Euclidean Overlap Metric [17] to quantify the distance between jobs. The Heterogeneous Euclidean Overlap Metric is a distance function, D(x, y), that can be used for both categorical and numeric features. Categorical features are assigned the minimal distance of 0 if they are identical, and the maximal distance 1 if they are different. For numeric features, the distance is defined as the difference between the two numeric features normalized by the range of values for that feature, yielding a distance between 0 and 1. We scaled the numeric features using the log-scale due to the large numeric ranges present in our dataset. Missing features are handled by returning a distance of 1 (i.e., a maximal distance). The total distance between two jobs is the square root of the squared sum of the individual feature distances.

We observed a large variance in job names in our dataset–only 40% of jobs appear to have run multiple times. One reason for this variance is that Hadoop assigns random job names to streaming jobs that allow users to run any executable or script as a mapper or reducer. Another reason is that users sometimes add unique identifiers to differentiate between jobs that are running the same application but that are processing different datasets. To cope with these variations and improve our ability to locate similar jobs, we computed the distance between two job names as the length of the longest common prefix divided by the maximum length of the job names.

6.2   Predicting Job Completion Times

We identified the K-Nearest-Neighbors for the incoming job based on the Heterogeneous Euclidean Overlap Metric, and predicted the job completion times using two algorithms namely: (i) the distance-weighted average algorithm [14], and (ii) the locally-weighted linear regression [15, 18] algorithm described below.

6.2.1   Distance-weighted average

The distance-weighted average predicts the job completion times using a Gaussian smoothing kernel [14, 18] function, $K(d) = e^{\frac{-d^2}{h}}$, to perform a distance-weighted average of the completion times of jobs in the neighborhood of the incoming job. The distance, d, is the Heterogeneous Euclidean Overlap Metric, and the bandwidth, h, is a smoothing factor. The Gaussian kernel function assigns a maximal weight of 1 to jobs that are identical to the incoming job, and lower weights to jobs that are further away. The kernel bandwidth modulates the weight assigned to jobs–smaller bandwidths assign larger weights to jobs with smaller distances.

6.2.2   Locally-weighted linear regression

The locally-weighted linear regression algorithm [15, 18] models job completion times as a multi-variate linear function of the features listed in 7. We used a weighted least-squares regression, which minimizes
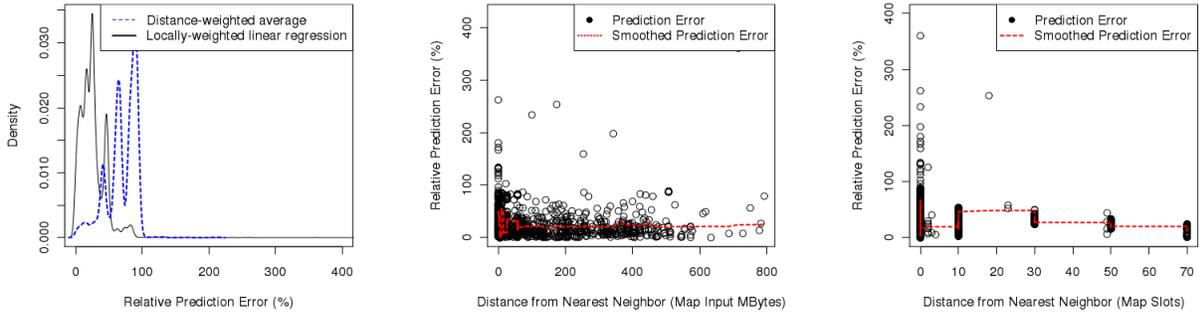
(a) Locally-weighted linear regression has a mean prediction error of ~26% compared to 70% for the distance-weighted algorithm.

(b) The mean prediction error of the linear regression model remains relatively constant as we scale the map input sizes.

(c) Large prediction errors were due to workload changes and performance problems.

Figure 7: Trends in the relative prediction error as we scale the map input sizes of three frequently-run jobs.

the weighted sum of squared residuals between the observed values and the predicted values. Weighting the residuals, emphasizes similar jobs, and de-emphasizes dissimilar jobs.

We measured the goodness of fit of our linear models using the adjusted coefficient of determination, R^2, which explains the percentage of variation that can be explained by the variables in our model. The adjusted R^2 adjusts for the number of parameters in the linear model and ranges from 0 to 1. A value of 1 indicates that the model perfectly explains the observed data. If we were unable to compute the adjusted R^2 value due to non-linear relationships in the data, we reverted to the distance-weighted average for our predictions.

We observed that some input features were linearly dependent, *i.e.*, collinear, in certain jobs. Linear regression in the presence of collinearity can lead to unstable estimates, *e.g.*, negative or very large estimates for the job completion times. We detected collinearity using the variance inflation factor[19] and sequentially dropped collinear variables. The variance inflation factor, $\frac{1}{1-R^2}$, is derived from the coefficient of determination, R^2, which is computed by performing a linear regression analysis of each independent variable using the remaining independent variables. Collinearity exists if the variance inflation factor exceeds 10.

6.3 Training

Our models need the following two parameters to be set: (i) the kernel bandwidth, H, which serves as a smoothing parameter for the distance-weighted average, and (ii) the size of the neighborhood, K. The size of the neighborhood should be large enough to generate robust linear models capable of accurately predicting job completion times, and yet small enough to exclude samples from dissimilar jobs which distort the outcome of the regression. We set the values to $H = 0.01$ for the distance-weighted average, $H = 1$ for the locally-weighted linear regression, and $K = 1000$ based on our experiments.

7 Prediction Results

We analyzed two aspects of our performance prediction algorithms namely: (i) the effect of scaling the input data size on the accuracy of our algorithms, and (ii) whether disequity in task durations could indicate

performance problems. We measured the accuracy of our algorithms using the relative prediction error, $\frac{abs(predicted-actual)}{actual}$, of job completion times.

7.1 Effect of scaling input data

We analyzed the performance of the distance-weighted algorithm and the locally-weighted linear regression algorithm at predicting job completion times when we scaled the map input sizes of three frequently-run jobs in our dataset. The training data consisted of 75,355 jobs whose map input sizes fell below the 75th percentile of map input sizes for each of the three job types. The test data consisted of 3,938 jobs whose map input sizes fell in the top-25th percentile.

Figure 7(a) shows that locally-weighted linear regression performs better with a mean relative prediction error of 26% compared to 70% for the distance-weighted algorithm. Figures 7(b) and 7(c) show the effect of scaling the map input sizes and map slots on the relative prediction error. We measured scale by computing the distance from the test jobs and their nearest neighbors in the training data.

We observed that the mean prediction error remained relatively constant as we scaled the map input sizes. However, contrary to our expectations, the largest variance in prediction errors occurred when the distance from the nearest neighbor was smallest. A manual inspection of the top-20 jobs with the largest prediction errors yielded the following sources of error:

– *Performance problems:* Performance problems due to application hangs, process failures, and a configuration problem which introduced a random 3-minute delay when instantiating sockets [11] led to large prediction errors in 35% of the jobs. An additional 15% of jobs exhibited large prediction errors due to performance problems in jobs in the training data which skewed the results. Outlier filtering might reduce these errors.

– *Workload changes:* 40% of the large prediction errors were due to workload changes where different application-specific input arguments led to large variations in the job completion times. Our algorithms were unable to detect these changes as we restricted them to the generic Hadoop-specific input parameters.

– *Poor linear models:* We hypothesize that the errors in 10% of the jobs might be due to non-linear behavior that we did not capture in our models.

7.2 Effect of task disequity

We used the locally-weighted linear regression model to predict the job completion times of successful, failed, and cancelled jobs for the entire dataset (see Figure 8(a)). The mean relative prediction error was high: 236% for successful jobs, 1215% for failed jobs, and 888% for cancelled jobs. Based on our analysis of error sources in Section 7.1, we hypothesize that the large prediction errors are due to performance problems, workload changes and poor linear models. When we restrict our results to the 80th percentile of prediction errors, the mean relative prediction error drops to 32% for successful jobs, 60% for failed jobs, and 74% for incomplete jobs indicating that the algorithm is good at detecting failures.

We used the Gini Coefficient, described in Section 5.3.1, to indicate the extent to which task durations are comparable across peers. The Gini Coefficient ranges from 0.0 to 1.0, and higher values indicate greater disequity among the durations of tasks. Figures 8(b) and 8(c) show the effect of disequity in the Gini Coefficient on the relative prediction error for the 80th percentile of jobs. Large Gini Coefficients in the reduce durations are correlated with large prediction errors implying that peer-comparison might be a feasible strategy for detecting performance problems and data skews.
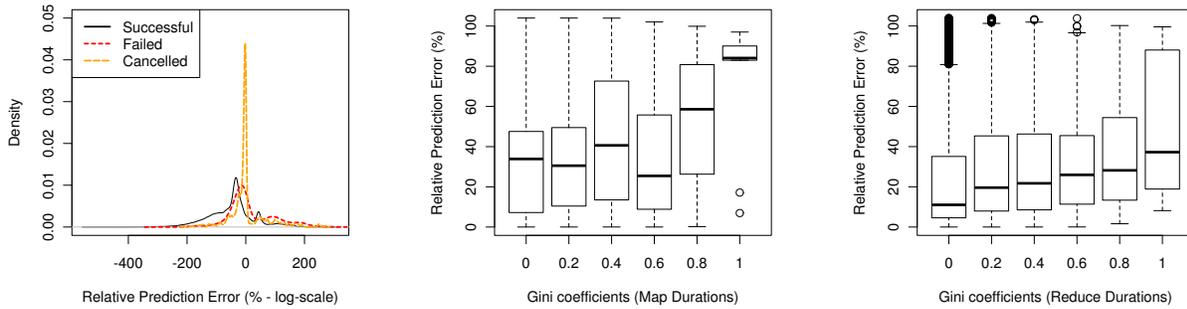
(a) The mean prediction error for successful jobs was lower than for failed and cancelled jobs indicating that locally-weighted linear regression is good at detecting failures.

(b) There is no pronounced trend in the effect of Gini Coefficients of map durations on the relative prediction error.

(c) Large Gini Coefficients in reduce durations are correlated with large prediction errors implying that peer-comparison is a feasible strategy for detecting performance problems.

Figure 8: Relative mean prediction errors for locally-weighted linear regression on entire dataset.

8   Related Work

Several studies [14, 15, 20] have applied instance-based learning approaches to predict the performance of job completion times in grid environments. Our study uses a similar approach to predict job completion times in MapReduce workloads, and outlines the reasons for the errors we observe.

Our analysis of MapReduce workloads corroborates observations from previous studies of workloads on parallel computer systems [16, 21, 22]. These studies observed that the job durations and inter-arrival times followed heavy-tailed distributions such as the log-uniform and Gamma distributions. Li et al. [22] and Downey et al. [16] also show that user behavior is uniform and predictable over short time intervals allowing better predictions to be made for improving the job scheduling.

Other studies of production traces of computer systems and clusters have also focused on failure data and characteristics [23, 24, 25, 26, 27]. All these studies focused on hardware failures, except [25], which examined component and service failures. [24, 26] examined specialized computer systems such as single-fault-tolerant Tandem computers and mainframes, while more recent studies examined High-Performance Computing (HPC) clusters, and web-service/Internet architectures. Of these studies, only [26] explicitly considered workload in concert with the failure data. Also, these studies did not examine specific workloads. The significant distinguishing features of our analysis is that it provides insight on job and user characteristics specific to the MapReduce workload, which is an important class of workloads in today's Cloud Computing.

Wang et al. [28] built a simulator for MapReduce workloads which captures predicts the expected application performance for varied MapReduce configurations. Their approach achieved high prediction accuracy, ranging between 3.42% and 19.32%. However, they require precise information on the compute cycles spent per input byte for each application–this information might not always be available. Instance-based learning approaches attempt to estimate job completion times even in the presence of uncertainty.

9   Conclusion

We analyzed Hadoop logs from the 400-node M45 [3] supercomputing cluster which Yahoo made freely available to select universities for systems research. Our studies tracks the evolution in cluster utilization

patterns from its launch at Carnegie Mellon University in April 2008 to April 2009. Job completion times and cluster allocation patterns followed a long-tailed distribution motivating the need for fair job schedulers [8] to prevent large jobs or heavy users from monopolizing the cluster. We also observed large error-latencies in some long-running tasks indicating that better diagnosis and recovery approaches are needed.

User tended to run the same job repeatedly over short intervals of time thereby allowing us to exploit temporal locality to predict job completion times. We compared the effectiveness of a distance-weighted algorithm against a locally-weighted linear algorithm at predicting job completion times when we scaled the map input sizes of incoming jobs. Locally-weighted linear regression performs better with a mean relative prediction error of 26% compared to 70% for the distance-weighted algorithm. We measured disequity in the distribution of task durations using the Gini Coeffient and observed that large Gini Coefficients in the reduce durations are correlated with large prediction errors implying that peer-comparison might be a feasible strategy for detecting performance problems.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters." *Communications of the ACM*, vol. 51, pp. 107–113, 2008.

[2] Hadoop, "Powered by Hadoop," http://wiki.apache.org/hadoop/PoweredBy.

[3] Yahoo!, "M45 supercomputing project," 2009, http://research.yahoo.com/node/1884.

[4] T. A. S. Foundation, "Hadoop," 2007, http://hadoop.apache.org/core.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *USENIX Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec. 2004, pp. 137–150.

[6] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System." in *ACM Symposium on Operating Systems Principles*, Lake George, NY, Oct 2003, pp. 29 – 43.

[7] B. Ripley, "R's MASS statistical package," 2009, http://cran.r-project.org/web/packages/VR/index.html.

[8] Yahoo!, "Hadoop capacity scheduler," 2008, https://issues.apache.org/jira/browse/HADOOP-3445.

[9] Hadoop, "DFSClient should invoke checksumok only once." 2008, https://issues.apache.org/jira/browse/HADOOP-4499.

[10] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, "SALSA: Analyzing Logs as State Machines," in *USENIX Workshop on Analysis of System Logs*, San Diego, CA, Dec. 2008.

[11] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Mochi: Visual Log-Analysis Based Tools for Debugging Hadoop," in *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, San Diego, CA, Jun. 2009.

[12] Hadoop, "Map/Reduce Tutorial," 2008, http://hadoop.apache.org/common/docs/current/mapred_tutorial.html.

[13] C. W. Gini, "Variability and mutability, contribution to the study of statistical distributions and relations," *Studi Economico-Giuridici della R. Universita de Cagliari*, 1912, reviewed in: Light, R.J., Margolin, B.H.: An Analysis of Variance for Categorical Data. *J. American Statistical Association*, Vol. 66 pp. 534-544 (1971).

[14] W. Smith, "Prediction services for distributed computing," in *International Parallel and Distributed Processing Symposium*, Long Beach, CA, March 2007, pp. 1–10.

[15] N. H. Kapadia, J. A. Fortes, and C. E. Brodley, "Predictive application-performance modeling in a computational grid environment," in *International Symposium on High-Performance Distributed Computing*. Redondo Beach, CA: IEEE Computer Society, Aug. 1999, p. 6.

[16] A. B. Downey and D. G. Feitelson, "The elusive goal of workload characterization," *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, vol. 26, no. 4, pp. 14–29, 1999.

[17] R. D. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *Journal of Artificial Intelligence Research*, vol. 6, pp. 1–34, 1997.

[18] S. S. Chris Atkeson, Andrew Moore, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, pp. 11–73, Apr. 1997.

[19] G. W. Stewart, "Collinearity and least squares regression," *Statistical Science*, vol. 2, no. 1, pp. 68–84, 1987. [Online]. Available: http://www.jstor.org/stable/2245615

[20] S. Krishnaswamy, S. W. Loke, and S. W. Loke, "Estimating computation times of data-intensive applications," *IEEE Distributed Systems Online*, vol. 5, no. 4, 2004.

[21] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *Journal of Parallel and Distributed Computing*, vol. 63, no. 11, pp. 1105–1122, Nov. 2003.

[22] H. Li, D. L. Groep, and L. Wolters, "Workload characteristics of a multi-cluster supercomputer," in *Job Scheduling Strategies for Parallel Processing*, New York, NY, Jun. 2004, pp. 176–193.

[23] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," in *Dependendable Systems and Networks*, Philadelphia, PA, Jun. 2006.

[24] J. Gray, "A census of tandem system availability between 1985 and 1990," in *IEEE Transactions on Reliability*, Oct. 1990.

[25] D. Oppenheimer, A. Ganapathi, and D. Patterson, "Why do Internet services fail, and what can be done about it?" in *USENIX Symposium on Internet Technologies and Systems*, Mar. 2003.

[26] R. Iyer, D. Rossetti, and M. Hsueh, "Measurement and modeling of computer reliability as affected by system activity," in *ACM Transactions on Computer Systems*, Aug. 1986.

[27] R. Sahoo, M. Squillante, A. Sivasubramaniam, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in *Dependendable Systems and Networks*, Florence, Italy, Jun. 2004.

[28] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in MapReduce setup," in *International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, London, UK, Sep. 2009.