INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2015; **28**:1265–1281
Published online 18 February 2014 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/dac.2759

Performance impact of JobTracker failure in Hadoop

Young-Pil Kim, Cheol-Ho Hong and Chuck Yoo*,†

Computer Science and Engineering, Korea University, Seoul, Korea

SUMMARY

In this paper, we analyze the performance impact of JobTracker failure in Hadoop. A JobTracker failure is a serious problem that affects the overall job processing performance. We describe the cause of failure and the system behaviors because of failed job processing in the Hadoop. On the basis of the analysis, we build a job completion time model that reflects failure effects. Our model is based on a stochastic process with a node crash probability. With our model, we run simulation of performance impact with very credible failure data available from USENIX called computer failure data repository that have been collected for past 9 years. The results show that the performance impact is very severe in that the job completion time increases about four times typically, and in a worst case, it increases up to 68 times. Copyright © 2014 John Wiley & Sons, Ltd.

Received 9 June 2013; Revised 11 October 2013; Accepted 21 January 2014

KEY WORDS: Hadoop; large-scale data processing; failure analysis; JobTracker

1. INTRODUCTION

Hadoop [1] is a popular open source implementation of Google's MapReduce [2] that is a data processing model for big data in large-scale cluster or cloud computing platform [3–5]. Hadoop is practical enough to be used in many commercial cloud systems such as Amazon EC2, Yahoo, and the Google search engine. Hadoop includes distributed file system (HDFS) for cloud storage systems. Thus, many cloud storage system researchers adopt Hadoop as their research platform because it is easy to analyze or modify.

The primary feature of Hadoop cluster is to maximize the job processing performance using parallelism. Basically, a Hadoop cluster is a master–worker structure; a master node splits a job into subtasks and assigns them to workers. Thus, a Hadoop cluster that consists of many nodes can execute many tasks concurrently. The job processing performance is determined by job completion time. Many studies [6, 7] have used the job completion time as an important metric for analyzing the performance of job processing in Hadoop cluster.

Another important feature of the Hadoop cluster is fault tolerance to recover data from node failures. The nodes of a Hadoop cluster are physical machines, so each node can crash because of various hardware problems such as overworked server, aging, and temperature [8, 9]. When a node crash happens, data and the job processing state can be corrupted or lost. A Hadoop cluster overcomes the node failures by redundancy. The data in Hadoop cluster spread redundantly in many nodes using DFS. Thus, the original data can be recovered by the redundant node even if the data in crashed node are unavailable. This approach requires an assumption that a master node is always fault-tolerant.

*Correspondence to: Chuck Yoo, Computer Science and Engineering, Korea University, Seoul 136713, Korea.
†E-mail: chuckyoo@os.korea.ac.kr

However, this assumption of fault-tolerance of Hadoop is not realistic because the master node can crash. For example, many recent studies [10–13] report that the master node crash is a single point of failure of Hadoop and should be handled. However, to the best of our knowledge, there has been no quantitative study as to the impact of master node crash. Recently, many studies [10–13] try to handle master node failure by replication of its metadata. But, the replication is not a perfect solution because of the high cost of replication [14]. (The more replicated master node is, the less node failure but the higher the cost is.)

In this paper, we analyze the performance impact of failure of JobTracker because master node crash causes JobTracker failure in Hadoop job processing. First, we review how Hadoop handles failures. Then, we build a job completion time model that reflects the JobTracker failure. Our model is based on a stochastic process with a node crash probability. And, we show the results of performance impact numerically with our model. As a result, we find that the impact of the failure of the JobTracker in Hadoop is much worse than expected because it increases the overall job execution time by 68 times in the worst case.

We summarize our contributions as follows. First, we extend the performance analysis for Hadoop. Previous study [14] for analysis of performance impact of Hadoop failure covers only nonmaster node such as TaskTracker and DataNode. Our work uncovers the performance impact of the master node such as JobTracker that the previous work did not. Second, our work urges that more elaborate solutions are needed for handling Hadoop failures. Recently, Hadoop supports high availability feature that replicates the master node. However, it still depends on replication approach that is not a perfect solution. Thus, we stress that the current Hadoop needs more solutions for failure handling.

The rest of this paper is organized as follows. Section 2 describes previous studies related to our work. Section 3 explores the problem and reviews Hadoop architecture. Section 4 describes the job completion time model, which includes the performance penalty because of a node crash. Section 5 shows the results of numerical analysis for the performance penalty in the various unreliable situations. In Section 6, we try to validate our model using well-known Hadoop simulator, MRPerf, and we provide the results of simulation and our model. Finally, we conclude in Section 7.

2. RELATED WORK

Traditionally, Hadoop adopts MapReduce [2] for a fault-tolerant data processing model. The model requires the assumption that master node is not faulty. However, the assumption is not realistic. There are many Hadoop users who use commodity hardware [15], and the commodity hardware is failure prone. Thus, the master node also can crash. Many studies [10–13] and reports [15–17] consider NameNode as a single point of failure that should be reliable and fault-tolerant. Google File System (GFS) [10] and UpRight-HDFS [11] use checkpoint-based replication method. To reduce the cost of replication, Wang's work [12] keeps important metadata only. To avoid a single point of failure, Maresia [13] uses a peer-to-peer model instead of master–worker architecture. To overcome the crash of NameNode, MapR [18] and Kuromatsu's work [19] uses automatic recovery method. Apache Hadoop [20] enhances the reliability of Hadoop distribution by introducing the automatic recovery and replication feature; however, the alpha version (later 2.0) is not sufficiently evaluated yet in research. The stable Hadoop version is 1.1.2, and it is based on 0.20.x, which is our target. The version is popular and used well in public and many recent studies [13, 14, 19, 21]; thus, it remains a JobTracker failure issue yet.

For nonmaster node failures, recent work [14] is published, and the authors suggest some discussion points, and the main thing is that delayed speculative execution that is Hadoop's execution model has a general problem, and we need more analysis works on Hadoop-like framework.

Several studies [6, 22, 23] have focused on characterizing and collecting failure data in large-scale computing systems. In Sahoo's work [22], the authors collected event logs of 395 IBM servers, analyzed node failure patterns, and produced probability distribution of daily failures. In Schroeder's work [23], the authors collected, analyzed 9-year failure data of 4750 machines at Los Alamos

DOI: 10.1002/dac

National Laboratory, and produced statistical properties. In Kavulya's work [6], the authors collected, analyzed 10-month data of 400 Hadoop cluster nodes, and predicted job completion time using linear regression. These previous studies accomplished the analysis of failure characteristic that given failure data follow a specific probabilistic distribution; however, they did not cover the master node crash impact on performance.

In contrast to the previous studies, out failure analysis is based on a very credible failure data available from USENIX. It is called the computer failure data repository (USENIX CFDR) [24]. The CFDR [24] started as an initiative at Carnegie Mellon University (CMU) in 2006 and was motivated by the fact that hardly any failure data from real large-scale production systems were available to researchers. There are 13 public data sets in CFDR (in 2012 years), and we use the Los Alamos National Lab (LANL) data set because LANL data are the largest failure data set and have the longest collection time. The data were collected over 9 years, cover more than 23,000 outages, and were the first to become publicly available as part of the CFDR.

3. BACKGROUND OF FAILURES IN HADOOP

Hadoop is an open source implementation of MapReduce data processing model [2]. The MapReduce enables distributed, data-intensive, and parallel applications by decomposition. The MapReduce decomposes a massive job into smaller tasks (map tasks and reduce tasks), and massive data of job are split into smaller partitions. Each decomposed task processes a different partition in parallel. A job in Hadoop consists of a group of map and reduce tasks. A map task executes a user-defined map function for each key/value pair in its input. A reduce task consists of a shuffle, sort, and reduce phase. During the shuffle and sort phase, the reduce task fetches, merges, and sorts the output 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 limits the maximum number of running tasks in a node by task slot specified in the system configuration. Hadoop shares data among distributed tasks in the system through the HDFS. HDFS splits and stores files as fixed-size blocks.

Hadoop uses a master–worker architecture with a master node and multiple worker nodes, as shown in Figure 1. The master node typically runs two daemons: the JobTracker that schedules and manages all of the tasks belonging to a running job and the NameNode that manages the HDFS namespace by providing a filename-to-block mapping and regulates access to files by tasks. Each worker node runs two daemons: the TaskTracker that launches tasks on its local node and tracks the progress of each task on its node and the DataNode that serves data blocks (on its local disks) to HDFS clients.

3.1. Failure handling in Hadoop job processing

To analyze the failure characteristic of Hadoop cluster, we need to know what failure occurs in Hadoop and how Hadoop handles the failures in the job processing. In this paper, the failure means the stop of normal program execution before its completion such as abort, crash, and hang. We classify the failures by the source of failures where the failures occur.

The first source is a task. Hadoop tasks (map tasks and reduce tasks) are Java applications, and they run in JVM. In Java application, a user-level exception causes abort and crash; thus, Hadoop tasks also may suffer the unhandled exception. TaskTracker notices that the task crash and marks the task execution as failed. Another case, a task can hang. The detection of hanging task is based on a timeout. When TaskTracker notices that it has not received a progress report for a while, it proceeds to mark the task as failed. All failed tasks are automatically killed after the timeout period.

The second source of failure is TaskTracker. The TaskTracker reports the progress status of underlying tasks. When the TaskTracker crashes, the JobTracker notices the crash by the loss of heartbeat. Then, the JobTracker excludes the troubled TaskTracker and respawns tasks for resumed execution. The last source of failure is the JobTracker. This is the most critical failure because the JobTracker is a single point of failure in Hadoop. When the JobTracker crashes, task scheduling

DOI: 10.1002/dac

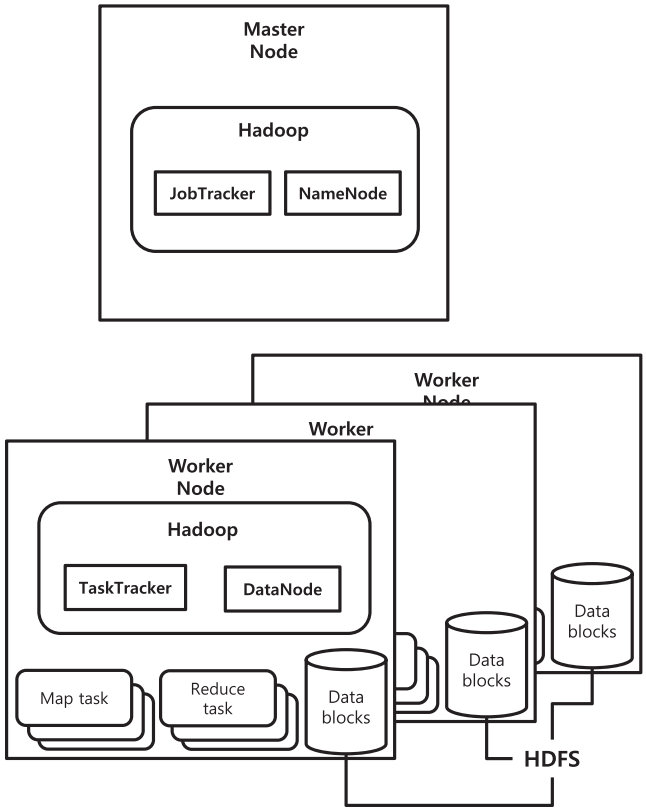

Figure 1. Hadoop cluster architecture.

Table I. Failures and the result in Hadoop job processing.

Source of failure	Failure cause	Results of failure	Loss
Task	Unhandled run-time user exception, crashing JVM, hanging task	Increasing a task execution time because of execution failing and restarting	Task
TaskTracker	Crashing TaskTracker process/node	Increasing a set of tasks execution time because of execution failing and restarting	Task set
JobTracker	Crashing JobTracker process/node	Losing a current job execution state and requiring manual restarting the job	Job

and all deployment operations are unavailable. Moreover, the state of the current job tasks is lost. In this case, the only way to resume execution is for a human administrator to restart the machine and reload the JobTracker. We summarize the aforementioned sources and failure handling of Hadoop in Table I. From Table I, we define the loss column for describing the severity of the failure. The reasons why JobTracker crash is catastrophic are two. First, the loss of failure is huge (an entire job). Second, the failure is not recoverable automatically by the Hadoop. (other failures can be recovered by the Hadoop.) Thus, this paper focuses on JobTracker failure.

DOI: 10.1002/dac

4. STOCHASTIC MODEL OF THE FAILURE OF HADOOP

4.1. Preliminaries

This section describes definitions of model parameters for modeling Hadoop job processing. We define a set of nodes to $N = \{N_1, N_2, \ldots, N_{|N|}\}$. The total number of nodes is $|N|$. The node set consists of a JobTracker and DataNodes. The number of task slots for node N_i is $|N_i|$. A task set that is managed by a JobTracker is $T = \{T_1, T_2, \ldots, T_{|T|}\}$, and the total number of tasks is $|T|$. Each task maps to a node task slot, and we call this a task execution. We focus on time to first node crash during a task execution. However, we do not know the exact time to first node crash. Thus, we define time to first crash as a random variable, X. A probability distribution function of X is denoted by $F_X(x)$. We assume that all node crash probabilities are independent to each other. This assumption is reasonable because each node is physically separated. The independence of each node means that the node failure occurred in each task does not propagate to other nodes. The execution time of a job is determined by the heaviest task. Without node failures, the heaviest task is purely determined by the computation requirement. However, when node failures occur, the heaviest task can be changed to another task on the failed node. This change is possible because of the extended execution time of tasks on the failed node from restarting and resuming the tasks. After a node crash, rebooting the node and recovering the node state are required. We define the time for rebooting and recovering the node a constant γ_R. When a task is failed or finished, a state for management of the task should be cleaned up for next execution. We define the clean-up time a constant γ_C.

There is a job computation requirement (ω). This is a pure computation time for a job processing. Then, the actual execution time of ω is $\tau(\omega)$. If there are no node crashes, $\tau(\omega) = \omega$. If there are some node crashes, $\tau(\omega)$ can be variable. In particular, we define a computation requirement for a JobTracker ω_J and execution time for a JobTracker $\tau(\omega_J)$.

4.2. Expected job completion time

Our modeling goal is to obtain expected job completion time that reflects a node crash probability, namely, $E[\tau(\omega)]$. Figure 2 depicts the computation requirement and those flows in cluster that consists of k DataNodes. Because a job is split into parallel execution units, the amount of computation requirement ω also is split into $\frac{\omega}{|T|} \cdot |N_i|$ where $\forall i, N_i \in N$. The computation requirement of JobTracker ω_J is dependent of the overall job completion because the JobTracker should wait until all task executions are finished to return final output result to a job client.

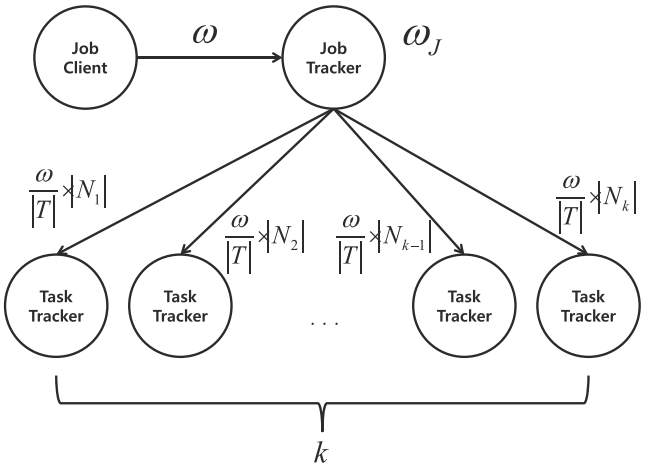

Figure 2. Cluster model and computation requirements.

DOI: 10.1002/dac

In a parallel execution environment, the total completion time is controlled by a task that has the longest execution time. This requires an assumption that all tasks start simultaneously, but the difference of starting time can be ignored if the task execution time is sufficiently long. Typical job in the cloud computing is heavy workload, so the long task execution time is reasonable and common. On the basis of the assumption, we derive the expected job completion time as follows.

Theorem 1 (Expected job completion time)
The expected job completion time of given job completion time requirement ω is as follows.

$$E[\tau(\omega)] = \max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\} + \frac{(\gamma_R+\gamma_C)F_X\left(\max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\}\right)}{1-F_X\left(\max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\}\right)} + \frac{\int_0^{\max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\}} x dF_X(x)}{1-F_X\left(\max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\}\right)}$$

We can prove the Theorem 1 by the following Lemma 1. The details of proof are described in Appendix A and B.

Lemma 1 (Expected subjob completion time)

$$E[\tau(\omega_i)] = \omega_i + \frac{\gamma_R F_X(\omega_i)}{1-F_X(\omega_i)} + \frac{\int_0^{\omega_i} x dF_X(x)}{1-F_X(\omega_i)}$$

4.3. Node crash probability model

The $F_X(x)$ of our model is a cumulative distribution function (CDF) of time to crash. In Section 4.2, we suggest the function as the general form because we do not need to know the exact numerical results in a model developing phase. For numerical evaluation, however, we need to define it.

We select the Weibull model for CDF. The reason is two. First, the Weibull model is popular and well used for developing failure theory model and analyzing failure data [25, 26]. Second, it is reported that the Weibull model fits best for representing failure data distribution in cluster environment [27].

For using the Weibull model, we need to determine two parameters: shape and scale parameter. Generally, the CDF of Weibull model is given by

$$F_X(x;\theta,\lambda) = 1 - e^{-(x/\lambda)^\theta}$$

where x denotes a time variable, θ is the shape parameter, and λ is the scale parameter of the distribution. Both parameters are positive. A value of $\theta > 1$ indicates that the curve of distribution rapidly increases with time. The curve smoothly increases in the case of $0 < \theta < 1$. The scale parameter λ affects the distribution extent, so the larger the scale parameter, the more spread out the distribution. The shape parameter θ affects the shape of the distribution, so the larger the scale parameter, the slower the curve of distribution.

For determining parameters of the Weibull model, we analyze failure data of LANL cluster at CFDR [28]. The data spans 22 high-performance computing systems that have been in production use at LANL between 1996 and November 2005 (9 years). The workloads run on those systems are large-scale long-running three-dimensional scientific simulations. These applications perform long periods (often months) of CPU computation, interrupted every few hours by a few minutes of I/O for checkpointing.

The failure data contains several information including crash start times and its downtimes of 22 cluster systems. For each cluster system, we calculated mean time to failure (MTTF) for evaluating system reliability. Generally, MTTF is well used for characterizing system reliability and the higher is more reliable. From the failure data, we exclude cluster data that has eight or fewer nodes because

DOI: 10.1002/dac

most Hadoop clusters have eight or more nodes. Also, we collect the failure events related to hardware failure because we consider node crash due to hardware failures. Then, we sorted MTTFs of the cluster systems and selected four cluster systems according to their MTTFs: highest MTTF, median MTTF, average MTTF, and lowest MTTF. The MTTF data of selected cluster systems are shown in the following Figure 3. Additionally, we describe the graphs of MTTF values and downtime values of all cluster systems in Appendix C.

In Figure 3, the *x*-axis means selected cluster systems, and the *y*-axis indicates minute times. The bar shows MTTF of selected cluster systems and the small number over each bar means the actual value of MTTF. We determined the Weibull parameters by applying probability plotting-based parameter estimation method [29] to the four selected data sets. First, we extracted cumulative histogram of hardware failure data of selected cluster systems. Second, for probability plotting, we found linear function versions of the Weibull curve. Third, we obtained the shape parameters by the slopes of the linear functions and the scale parameters also obtained from the functions. The detailed steps are shown in [29]. The following Table II shows the Weibull parameters that we found.

From Table II, we can depict the Weibull CDF graphs in Figure 4. The graph is CDF, so the characteristics of distributions of time-to-crash values are shown. The 99% of time-to-crash values

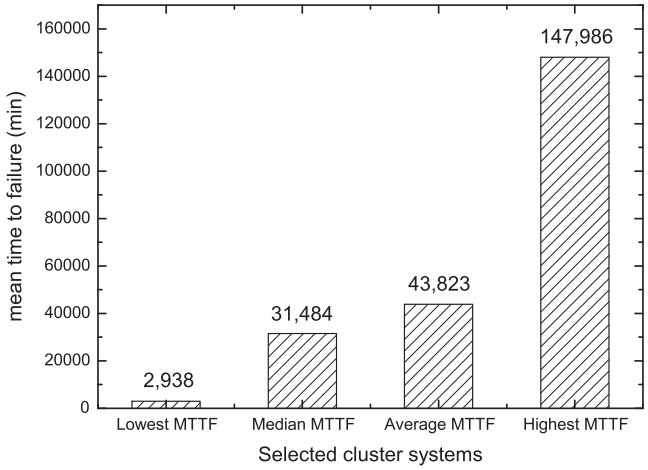

Figure 3. Mean time to failures (MTTFs) of selected cluster systems.

Table II. Weibull models for node crash.

Model	Weibull function parameters	MTTF
W1	$F_X(x;\theta = 0.438, \lambda = 1115.3)$	Lowest MTTF (2,938), unreliable
W2	$F_X(x;\theta = 0.590, \lambda = 28487.8)$	Median MTTF (43,823), unstable
W3	$F_X(x;\theta = 0.892, \lambda = 29775)$	Average MTTF (31,484), unstable
W4	$F_X(x;\theta = 1.031, \lambda = 149875)$	Highest MTTF (147,986), reliable

MTTF, mean time to failure.

DOI: 10.1002/dac

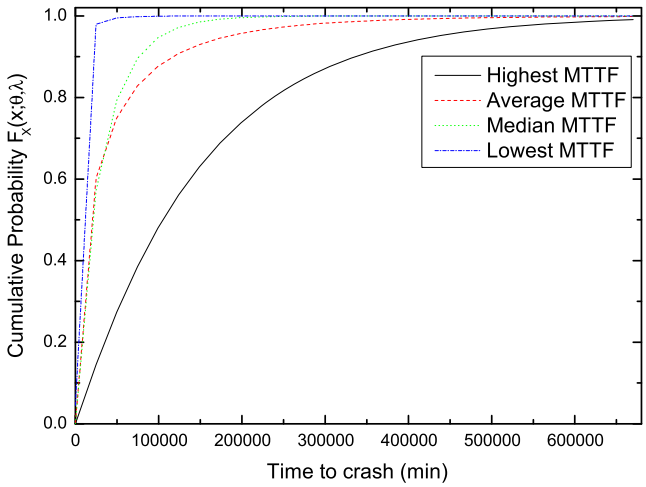

Figure 4. Weibull cumulative distribution curves for selected cluster systems.

in 'lowest MTTF' cluster are under 42.9 K mins, so the cluster easily crashes. However, 'highest MTTF' cluster has many relatively long time-to-crash values (99% under 670 K mins), so the cluster more stable than the 'lowest MTTF' cluster.

The graphs represent the probabilistic distribution models of time to crash, and the models are based on the real data. Thus, they are reasonable. In Section 5, we use the failure distribution models for numerical analysis.

5. NUMERICAL RESULTS

Our goal is to examine the variation of expected job completion time in Hadoop when a node crash probability is given. In particular, we focus on the overhead, which means the increased expected job completion time because of a JobTracker node crash. To that end, we consider three cases: the worst case, where the failure distribution of a JobTracker node follows the model W1; the median case, where the failure distribution of a JobTracker node follows the model W2; and the average case, where the failure distribution of a JobTracker node follows the model W3. We exclude the best case, where the failure distribution of a JobTracker node follows the model W4, because that is meaningless in terms of overhead. For each case, we assume that DataNodes have the node crash probabilities listed in Table II (W1, W2, W3, and W4) because these reflect the general node failure characteristics of the actual cluster environment.

- Case 1. 'Highest JobTracker node crash probability' (W1)
- Case 2. 'Median JobTracker node crash probability' (W2)
- Case 3. 'Average JobTracker node crash probability' (W3)

To obtain numerical results, we need to select the following additional parameters (Section 4.1): the job computation requirement (ω) and the cluster configuration ($|T|, |N|, |N_i|, \gamma_R, \gamma_C$). First, we define three job computation requirements: 'an-hour-job' ($\omega = 60$ min), 'a-day-job' ($\omega = 1440$ min), and 'a-month-job' ($\omega = 43200$ min). We assume that the jobs keep running until their job computation requirements are completely satisfied. These three job computation requirements are common in a large-scale cluster system. Second, we find the parameter values of the cluster configuration from the real failure data [28]. Actually, the failure data do not include recovery time (γ_R) and cleanup time (γ_C) explicitly, so we use the average downtime of each cluster system. Because the downtime is the elapsed time between a node crash and re-execution, we can use that as γ_R. For the cleanup time, γ_C, we assume a fixed value of 2 min because we think a node that receives cleanup request from a JobTracker can be rebooted within that time.

DOI: 10.1002/dac

Additionally, the cluster configuration parameters are related to the Weibull model, because each Weibull model represents the failure characteristic of each cluster.

The extracted configuration parameters and the corresponding Weibull parameters are listed in Table III.

Note that the values of γ_R are somewhat larger than our job computation requirement per node (see $\omega \times \frac{|N_i|}{|T|}$ in Section 4.2). For example, an-hour-job in the lowest MTTF cluster has 0.06 min $\left(\omega \times \frac{|N_i|}{|T|} = 60 \times \frac{4}{4096}\right)$, and a-day-job in the average MTTF cluster has 5.63 min $\left(\omega \times \frac{|N_i|}{|T|} = 1440 \times \frac{2}{512}\right)$. This means that the γ_R value can affect the numerical result of the equation in Theorem 1. Thus, we consider the effect of γ_R for two extremes: minimum cost ($\gamma_R = 100.94$ min) and maximum cost ($\gamma_R = 2540.79$ min). Note that downtime can be large if the recovery time is long, although it has the average MTTF like our average MTTF cluster because downtime depends on recovery time, and MTTF depends on uptime.

For these two γ_R values, we calculate the expected job completion time of the JobTracker ($E[\tau(\omega_J)]$) and the maximum expected job completion time of a DataNode ($\max\{E[\tau(\omega_i)], \forall i, N_i \in N\}$). Then, we obtain the additional job completion time overhead as

$$\frac{E[\tau(\omega_J)]}{\max\{E[\tau(\omega_i)], \forall i, N_i \in N\}} - 1 \tag{1}$$

The first case is that of 'high JobTracker crash probability'. In this case, the JobTracker runs in the cluster of lowest MTTF and follows the Weibull model W1. Other DataNodes run in the clusters of highest MTTF,average MTTF,median MTTF, and lowest MTTF. Figure 5 shows the results for the additional job completion time overhead.

Table III. The values of cluster configuration parameters.

Clusters	Cluster configuration parameters					Weibull models
	$\|T\|$	$\|N\|$	$\|N_i\|$	γ_R (min)	γ_C (min)	
Lowest MTTF	4096	1024	4	168.3	2	W1
Median MTTF	512	128	4	142.07	2	W2
Average MTTF	512	256	2	2540.79	2	W3
Highest MTTF	128	32	4	100.94	2	W4

MTTF, mean time to failure.

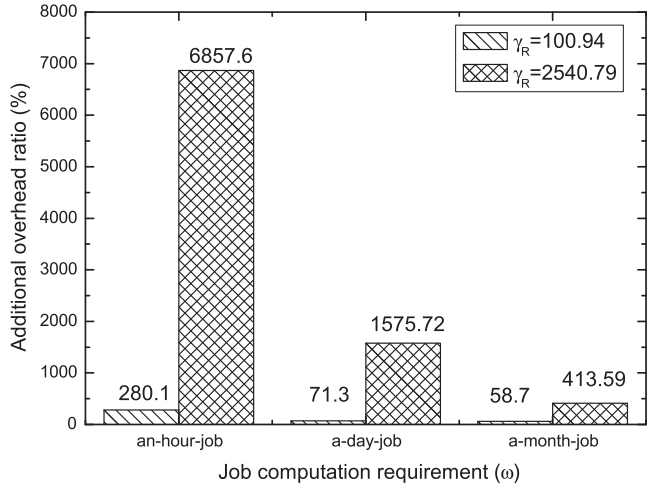

Figure 5. Results for additional job completion time overhead of high JobTracker crash probability.

DOI: 10.1002/dac

The *x*-axis describes the job computation requirement: an-hour-job, a-day-job, and a-month-job. The *y*-axis is the additional job completion time overhead (Equation (1)), and this value is relative to the job requirement time. If the additional job completion time overhead is zero, the JobTracker does not affect the job completion time at all. We also consider the recovery cost of the JobTracker. Thus, by referring to Table III, we apply two recovery costs: minimum recovery cost ($\gamma_R = 100.94$ min) and maximum recovery cost ($\gamma_R = 2540.79$ min).

The results in Figure 5 show that the expected job completion time is greatly increased for all job computation requirements. This means that the JobTracker significantly affects the job completion time. In particular, the results show that the job completion time increases when the recovery cost increases. The recovery cost is due to the JobTracker, so this implies that the recovery mechanism of the JobTracker should be efficient.

Although, in Figure 5, the overhead shown for an-hour-job is the highest, its actual value is small. This can be seen in Figure 6.

Figure 6 shows the actual value increments for an-hour-job, a-day-job, and a-month-job (zero means no increment). We use the log scale in the *y*-axis of the graph in Figure 6 for easy presentation. Figure 5 shows that an-hour-job with '$\gamma_R = 2540.79$ min' reaches 6857.6%, which it is greater than the 1575.72% of a-day-job with $\gamma_R = 2540.79$ min. However, the actual value (190.6 min) for an-hour-job is smaller than that (709.5 min) for a-day-job.

Above all, in our example of highest JobTracker crash probability, the additional job completion time overhead ranges from 58.7% to 6857.6%. The overhead of 6857.6% is not exaggerated, because these values are based on actual failure data [28]. Thus, we can say that these results are reasonable.

The second case is that of 'median JobTracker crash probability'. In this case, the JobTracker runs in the cluster of median MTTF and follows the Weibull model W2. Other DataNodes run in the clusters of highest MTTF,average MTTF,median MTTF, and lowest MTTF. Figure 7 shows the results for the additional job completion time overhead.

The *x*-axis and *y*-axis are the same as those in Figure 5. Likewise, we consider the recovery cost as with the highest JobTracker crash probability. Figure 7 shows again that the JobTracker affects the job completion time (maximum 394.68%). Note that this JobTracker runs in the median crash probability environment, which means that the result is based on the median MTTF.

The third case is that of the 'average JobTracker crash probability'. In this case, the JobTracker runs in the cluster of average MTTF and follows the Weibull model W3. Other DataNodes run in the clusters of highest MTTF,average MTTF,median MTTF, and lowest MTTF. Figure 8 shows the results for the additional job completion time overhead.

The *x*-axis and *y*-axis are the same as those in Figure 5. Likewise, we consider the recovery cost as with the highest JobTracker crash probability. Figure 8 shows again that the JobTracker affects

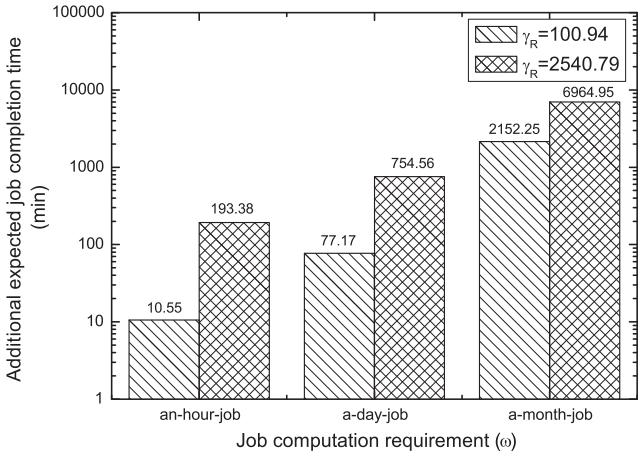

Figure 6. Expected job completion time increment.

DOI: 10.1002/dac

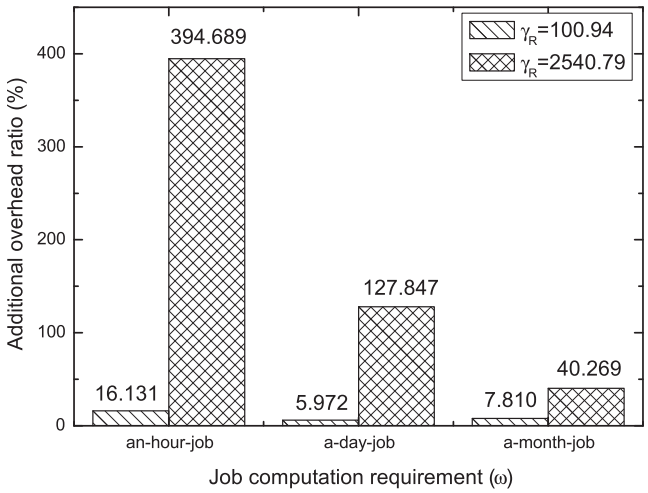

Figure 7. Results for additional job completion time overhead of median JobTracker crash probability.

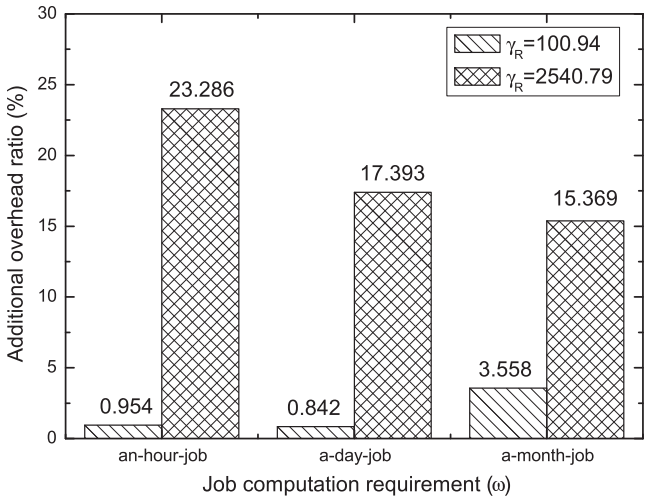

Figure 8. Results for additional job completion time overhead of average JobTracker crash probability.

the job completion time (maximum 23.28%). Note that this JobTracker runs in the average crash probability environment, which means that the result is based on the average MTTF. Because the result is based on average value, we can say that it is representative.

The average MTTF is an average of the MTTFs of all cluster systems represented by the given failure data. There is no system that has the average MTTF, because it is a representative value. However, one of the given cluster systems actually has the median MTTF. Thus, we believe that the median result is much more practical than the average result.

On the basis of the aforementioned results, we are convinced that the JobTracker with node crash probability should be a primary target of failure management for reliable large-scale parallel data processing. Moreover, we can say that reducing the recovery cost of the JobTracker is required, because we see from all the results that its recovery cost largely affects the job completion time increment.

Currently, the Hadoop depends on the checkpoint-based recovery mechanism; thus, the recovery cost is not inevitable. Note that from our results, we showed previously that the job completion time can have 6857.6% of additional overhead in the worst case and 394.68% in a typical case. These results are very serious and indicate that the recovery paradigm should be improved or changed.

DOI: 10.1002/dac

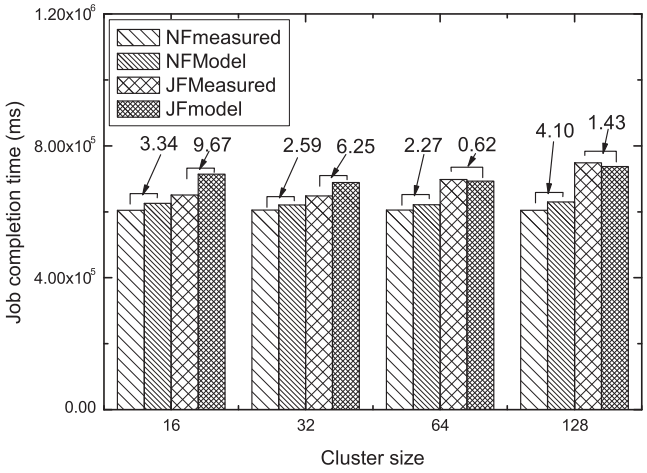

Figure 9. Job completion time of simulation and model.

6. MODEL VALIDATION

For model validation, we use MRPerf [30–32]. MRPerf is a Hadoop simulator widely used in many Hadoop studies [33–35] and is based on the NS-2 network simulator. MRPerf is a fairly accurate simulator because it predicts Hadoop performance within a range of 5.22–12.83% of the actual measurements in real Hadoop clusters [31]. MRPerf is valid for clusters with 16–128 nodes. Many previous Hadoop studies, which utilize real Hadoop clusters, used 128 nodes or less. For example, Kavulya's work [6] used 27 nodes, and others [11, 19, 21] used 5 or 50 nodes. For our research, we conduct several experiments using clusters of 16, 32, 64, and 128 nodes to validate our model. We use TeraSort as the default workload of the simulator. For each cluster, we measure job completion time in two failure cases: a node failure and a JobTracker failure. For our model, we calculate the model value of a node failure using Lemma 1 and the model value of a JobTracker failure using Theorem 1. We then compared the measured values with the model values. The results are in Figure 9.

In Figure 9, the *x*-axis indicates cluster sizes, and the *y*-axis indicates job completion time. 'NF measured' and 'JF measured' stand for the simulation results of node failure and JobTracker failure, respectively. Similarly, 'NF model' and 'JF model' are the expected job completion times for node failure and JobTracker failure in the model. The numbers with arrows (above the bar graphs) are the model errors in terms of percentages. For example, 3.34 and 9.67 on the 16 nodes cluster indicate that the value of node failure model is 3.34% larger than its measured value, and the value of JobTracker model is 9.67% larger than its measured value. From Figure 9, we can see that the results of simulation are similar to the results of our model because all model errors are smaller than 10%. In fact, our model error range (from 0.62% to 9.67%) is better than that of MRPerf (simulation error range from 5.22% to 12.83%), which means that our model is as accurate as MRPerf.

7. CONCLUSION

In this paper, we describe and analyze the performance impact of JobTracker failure in Hadoop. As a result, we find that the failure of the JobTracker is critical, because it seriously affects the overall job execution time. Quantitatively, we find that in the worst case, the job execution time is about 68 times more than its normal value for diverse node crash environments. Moreover, even in a typical case, the job completion time increases by about four times.

In our work, we cover the research area of performance analysis of JobTracker failure, which is caused by master node crash. This paper differs from the previous research that they do not analyze JobTracker failure. Moreover, the result of our work emphasizes the need of more elaborate

DOI: 10.1002/dac

solutions for handling JobTracker failures because the current Hadoop versions still suffer the huge performance degradation.

APPENDIX A: PROOF OF LEMMA 1

$$E[\tau(\omega_i)] = \omega_i + \frac{\gamma_R F_X(\omega_i)}{1 - F_X(\omega_i)} + \frac{\int_0^{\omega_i} x dF_X(x)}{1 - F_X(\omega_i)}$$

For our proof, we use the following axioms.

(a1) By the law of total expectation, the equation $E[\tau(\omega_i)] = E[E(\tau(\omega_i)|X = x)]$ holds true.
(a2) By the properties of CDF, the equations $\lim_{n\to\infty} F_X(n) = 1$ holds true.
(a3) We do not consider an always-crashed (time-to-crash value is 0) node; thus, $F_X(0) = 0$ holds true.

Proof

We start from Equation (2), and the equation looks like

$$E[\tau(\omega_i)|X = x] = \begin{cases} \omega_i & , x \geqslant \omega_i \\ x + \gamma_R + E[\tau(\omega_i)] & , x < \omega_i \end{cases}$$

The expectation of Equation (2) can be obtained as follows.

$E[E(\tau(\omega_i)|X = x)]$

$$= \int_0^{\infty} E(\tau(\omega_i)|X = x) dF_X(x)$$
$$= \int_0^{\omega_i} E(\tau(\omega_i)|X = x) dF_X(x) + \int_{\omega_i}^{\infty} E(\tau(\omega_i)|X = x) dF_X(x)$$

For convenience of derivation, we split the aforementioned equation into two integral equation parts: A and B. Namely, $E[E(\tau(\omega_i)|X = x)] = A + B$. We solve A and B in order.

$$\begin{aligned} A &= \int_0^{\omega_i} E(\tau(\omega_i)|X = x) dF_X(x) \\ &= \int_0^{\omega_i} (x + \gamma_R + E[\tau(\omega_i)]) dF_X(x) \\ &= \int_0^{\omega_i} x dF_X(x) + \int_0^{\omega_i} (\gamma_R + E[\tau(\omega_i)]) dF_X(x) \\ &= \int_0^{\omega_i} x dF_X(x) + (\gamma_R + E[\tau(\omega_i)]) \cdot (F_X(\omega_i) - F_X(0)) \end{aligned}$$

By the axiom (a3), $F_X(0) = 0$; thus, we get the following reduced equation.

$$= \int_0^{\omega_i} x dF_X(x) + \gamma_R \cdot F_X(\omega_i) + E[\tau(\omega_i)] \cdot F_X(\omega_i)$$

Now, we solve the equation B.

$$\begin{aligned} B &= \int_{\omega_i}^{\infty} E(\tau(\omega_i)|X = x) dF_X(x) \\ &= \lim_{n\to\infty} \int_{\omega_i}^{n} E(\tau(\omega_i)|X = x) dF_X(x) \\ &= \lim_{n\to\infty} (\omega_i \cdot F_X(n) - \omega_i \cdot F_X(\omega_i)) \\ &= \omega_i \cdot \lim_{n\to\infty} F_X(n) - \omega_i \cdot F_X(\omega_i) \end{aligned}$$

By the axiom (a2), $\lim_{n\to\infty} F_X(n) = 1$; thus, we reduce the aforementioned equation as follows.

$$= \omega_i - \omega_i \cdot F_X(\omega_i)$$

Now, we solve $E[E(\tau(\omega_i)|X = x)]$ using A and B as follows.

$$\begin{aligned} E[E(\tau(\omega_i)|X = x)] &= A + B \\ &= \left(\int_0^{\omega_i} x dF_X(x) + \gamma_R \cdot F_X(\omega_i) + E[\tau(\omega_i)] \cdot F_X(\omega_i)\right) + (\omega_i - \omega_i \cdot F_X(\omega_i)) \end{aligned}$$

DOI: 10.1002/dac

By the axiom (a1), we obtain the following equation.

$$E[\tau(\omega_i)] = \left(\int_0^{\omega_i} x dF_X(x) + \gamma_R \cdot F_X(\omega_i) + E[\tau(\omega_i)] \cdot F_X(\omega_i)\right) + (\omega_i - \omega_i \cdot F_X(\omega_i))$$

Then, we rearrange the above equation for $E[T(\omega_i)]$, and finally, Equation 4 is shown as follows.

$$E[\tau(\omega_i)] = \int_0^{\omega_i} x dF_X(x) + \gamma_R \cdot F_X(\omega_i) + E[\tau(\omega_i)] \cdot F_X(\omega_i) + \omega_i - \omega_i \cdot F_X(\omega_i)$$

$$E[\tau(\omega_i)] \cdot (1 - F_X(\omega_i)) = \int_0^{\omega_i} x dF_X(x) + \gamma_R \cdot F_X(\omega_i) + \omega_i \cdot (1 - F_X(\omega_i))$$

$$\begin{aligned} E[\tau(\omega_i)] &= \frac{\int_0^{\omega_i} x dF_X(x)}{(1-F_X(\omega_i))} + \frac{\gamma_R \cdot F_X(\omega_i)}{(1-F_X(\omega_i))} + \frac{\omega_i \cdot (1-F_X(\omega_i))}{(1-F_X(\omega_i))} \\ &= \frac{\int_0^{\omega_i} x dF_X(x)}{1-F_X(\omega_i)} + \frac{\gamma_R \cdot F_X(\omega_i)}{1-F_X(\omega_i)} + \omega_i \\ &= \omega_i + \frac{\gamma_R \cdot F_X(\omega_i)}{1-F_X(\omega_i)} + \frac{\int_0^{\omega_i} x dF_X(x)}{1-c(\omega_i)} \end{aligned}$$

□

APPENDIX B: PROOF OF THEOREM 1

The expected job completion time of given job completion time requirement ω is as follows.

$$E[\tau(\omega)] = \max\left\{E\left[\tau\left(\frac{\omega}{|T|} \cdot |N_i|\right)\right], \forall N_i \in N\right\} + \frac{(\gamma_R+\gamma_C) F_X\left(\max\left\{E\left[\tau\left(\frac{\omega}{|T|} \cdot |N_i|\right)\right], \forall N_i \in N\right\}\right)}{1-F_X\left(\max\left\{E\left[\tau\left(\frac{\omega}{|T|} \cdot |N_i|\right)\right], \forall N_i \in N\right\}\right)} + \frac{\int_0^{\max\left\{E\left[\tau\left(\frac{\omega}{|T|} \cdot |N_i|\right)\right], \forall N_i \in N\right\}} x dF_X(x)}{1-F_X\left(\max\left\{E\left[\tau\left(\frac{\omega}{|T|} \cdot |N_i|\right)\right], \forall N_i \in N\right\}\right)}$$

Proof

The requirement of computation amount for N_i is $\frac{\omega}{|T|} \cdot |N_i|$, and this is our new subjob. Therefore, the subjob completion time is $\tau\left(\frac{\omega}{|T|} \cdot |N_i|\right)$. If all subjobs run in parallel, and there is no node crash, the total job completion time can be obtained easily as

$$\tau(\omega) = \max\left\{\tau\left(\frac{\omega}{|T|} \cdot |N_i|\right), \forall N_i \in N\right\}$$

Now, our target is a subjob completion time per node, and we also have to consider the node crash. For notational convenience, we define the subjob completion time per node as

$$\tau(\omega_i) = \tau\left(\frac{\omega}{|T|} \cdot |N_i|\right) \tag{2}$$

When a node crash occurs, the crashed node should be rebooted and recovered; we already define the cost time γ_R. Now, we have the expectation of the subjob completion time of the DataNode crash when a node crash occurs. The result is as follows.

$$E[\tau(\omega_i)|X = x] = \begin{cases} \omega_i & , x \geq \omega_i \\ x + \gamma_R + E[\tau(\omega_i)] & , x < \omega_i \end{cases} \tag{3}$$

The expectation of subjob completion time increases when a first node crash occurs before the subjob is completed ($x < \omega_i$). From aforementioned equation (3), we can obtain the $E[\tau(\omega_i)]$ using the law of total expectation ($E(X) = E[E(X|Y)]$). Thus, we need to obtain the expectation of Equation (3) by Lemma 1. The derivation result is as follows.

$$E[\tau(\omega_i)] = \omega_i + \frac{\gamma_R F_X(\omega_i)}{1 - F_X(\omega_i)} + \frac{\int_0^{\omega_i} x dF_X(x)}{1 - F_X(\omega_i)} \tag{4}$$

DOI: 10.1002/dac

There are two types of node crash: a JobTracker node crash and a DataNode crash. First, we obtain the expectation of the subjob completion time when a DataNode crash occurs. For a DataNode crash, we can directly use the Equation (4). However, a JobTracker node crash is different. When a JobTracker node crash occurs, the node should be rebooted and recovered. In addition, the DataNodes should be cleaned up for reassignment of tasks. The computation requirement ω_J is also different with ω with because the JobTracker does not participate in a parallel computation of ω. Definitely, ω_J should be larger than the any other subjob completion time ($\omega_J \geqslant \mathrm{T}(\omega_i)$) because the JobTracker should wait until all task executions are finished. Note that these task execution times are stochastic, so we need their expectation values. Thus, we obtain

$$\omega_J \;=\; \max\{E[\tau(\omega_i)], \forall \omega_i\} \tag{5}$$

The JobTracker completion time is also affected by node crash probability. Thus, we have the expectation of the JobTracker completion time. We can derive the expectation of $\tau(\omega_J)$ based on the equation (3) as follows.

$$\begin{aligned} &E[\tau(\omega_J)|X \;=\; x] \\ &= \begin{cases} \omega_J & , x \geqslant \omega_J \\ x + \gamma_R + \gamma_C + E[\tau(\omega_J)] & , x < \omega_J \end{cases} \end{aligned} \tag{6}$$

From Equation (6), we have the following equation by the law of total expectation as the same way to Equation (4).

$$E[\tau(\omega_J)] \;=\; \omega_J + \frac{(\gamma_R + \gamma_C) F_X(\omega_J)}{1 - F_X(\omega_J)} + \frac{\int_0^{\omega_J} x dF_X(x)}{1 - F_X(\omega_J)} \tag{7}$$

Now, we obtain $E[\tau(\omega)]$. In a parallel execution environment, the job completion time is controlled by a node that has the longest subjob execution time. Thus, we have the following equation.

$$E[\tau(\omega)] \;=\; \max \left\{ \begin{array}{l} \max\{E[\tau(\omega_i)], \forall i, N_i \in N\}, \\ E[\tau(\omega_J)] \end{array} \right\} \tag{8}$$

Equation (8) can be reduced to $E[\tau(\omega_J)]$ because the JobTracker should wait until all DataNode executions are completed. Namely, $E[\tau(\omega_J)] \geqslant \max\{E[\tau(\omega_i)], \forall i, N_i \in N\}$. Thus, we obtain the following derivation result.

$$E[\tau(\omega)] \;=\; E[\tau(\omega_J)] \tag{9}$$

By the synthesis of Equations (2), (5), (7), and (9), we obtain the following expected job completion time.

$$\begin{aligned} E[\tau(\omega)] = \max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\} + \frac{(\gamma_R+\gamma_C) F_X\left(\max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\}\right)}{1-F_X\left(\max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\}\right)} \\ + \frac{\int_0^{\max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\}} x dF_X(x)}{1-F_X\left(\max\left\{E\left[\tau\left(\frac{\omega}{|T|}\cdot|N_i|\right)\right], \forall N_i \in N\right\}\right)} \end{aligned}$$

□

APPENDIX C: MEAN TIME TO FAILURE AND DOWNTIME VALUES OF THE LOS ALAMOS NATIONAL LAB CLUSTERS

We use the measured statistical data in the LANL trace data. The distribution of the MTTF values and the average downtime values of clusters are shown in Figure C.1 and C.2. In raw LANL data, there are a total of 23 clusters numbered from 2 to 24 ('cluster no.' in both figures). We exclude clusters with eight nodes or less (Section 4.3), and they are cluster nos 7, 15, 17, 22, 23, and 24. Overall, for all the data, the MTTF values of clusters range from 4119.56 to 18416.58 min, and their downtime values range from 100.94 to 9473.78 min.

DOI: 10.1002/dac

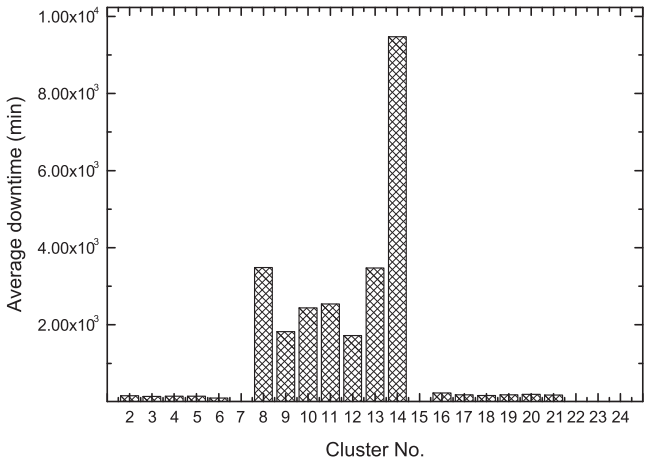

Figure C.1. Mean time to failure (MTTF) values of the Los Alamos National Lab (LANL) clusters.

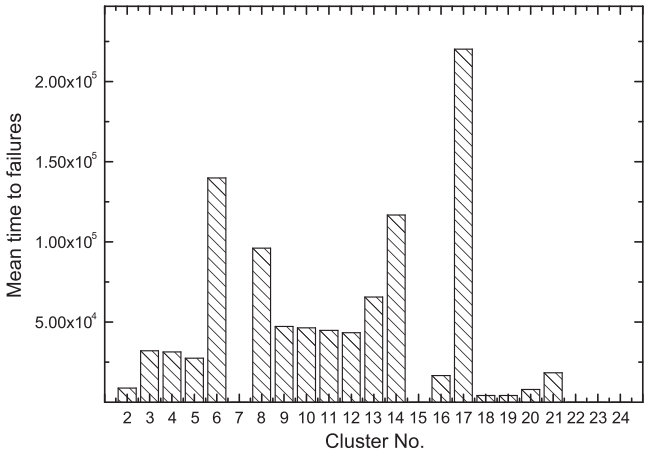

Figure C.2. Average downtime values of the Los Alamos National Lab (LANL) clusters.

In both Figures C.1 and C.2, the cluster number corresponding to the average MTTF cluster is 11. The average MTTF value of all clusters is 44836.81 min. In general, the MTTF value is different from the average downtime. For example, the MTTF value of cluster no. 14 is 116742.98 min, but its average downtime is 9473.78 min. Downtime value can be large if the recovery time from node failure is long.

ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea grant funded by the Korea government(MEST) (no. 2011-0029848).

REFERENCES

1. White T. *Hadoop: The Definitive Guide*. O'Reilly Media Inc. and Yahoo Press: Sebastopol, CA, USA, 2010.
2. Ghemawat S, Dean J. Mapreduce: Simplified data processing on large clusters. *Symposium on Operating System Design and Implementation (OSDI '04)*, San Francisco, CA, USA, 2004; 107–113.
3. Rimal BP, Choi E. A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing. *International Journal of Communication Systems* 2012; **25**(6):796–819.
4. Ahn S, Lee S, Yoo S, Park D, Kim D, Yoo C. Isolation schemes of virtual network platform for cloud computing. *KSII Transactions on Internet and Information Systems (TIIS)* 2012; **6**(11):2764–2783.
5. Hsu I-C. Multilayer context cloud framework for mobile Web 2.0: a proposed infrastructure. *International Journal of Communication Systems* 2013; **26**(5):610–625. DOI: 10.1002/dac.1365.

6. Kavulya S, Tan J, Gandhi R, Narasimhan P. An analysis of traces from a production mapreduce cluster. *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, Melbourne, Victoria, Australia, 2010; 94–103.
7. Chang H, Kodialam M, Kompella R, Lakshman T, Lee M, Mukherjee S. Scheduling in mapreduce-like systems for fast completion time. *Proceedings of IEEE INFOCOM*, Shanghai, China, 2011; 3074–3082.
8. Dean J. Evolution and future directions of large-scale storage and computation systems at google. *Proceedings of the 1st ACM Symposium on Cloud Computing*, Indianapolis, USA, 2010; 1.
9. Vishwanath KV, Nagappan N. Characterizing cloud computing hardware reliability. *Proceedings of the 1st ACM Symposium on Cloud Computing*, Indianapolis, USA, 2010; 193–204.
10. Ghemawat S, Gobioff H, Leung ST. The google file system. *ACM SIGOPS Operating Systems Review*, New York, USA, 2003; 29–43.
11. Clement A, Kapritsos M, Lee S, Wang Y, Alvisi L, Dahlin M, Riche T. Upright cluster services. *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, Big Sky, MT, USA, 2009; 277–290.
12. Wang F, Qiu J, Yang J, Dong B, Li X, Li Y. Hadoop high availability through metadata replication. *Proceedings of the First International Workshop on Cloud Data Management*, Hong Kong, China, 2009; 37–44.
13. Marcos PB. Maresia: an approach to deal with the single points of failure of the mapreduce model. *thesis for the degress of master of computer science*, 2013.
14. Dinu F, Ng T. Understanding the effects and implications of compute node related failures in hadoop. *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, Delft, The Netherlands, 2012; 187–198.
15. Shiran T. Top 10 namenode related problems. (Available from: http://www.mapr.com/blog/top-10-namenode-related-problems/) [Accessed, 2013].
16. Collins E. Apache hadoop availability. (Available from: http://blog.cloudera.com/blog/2011/02/hadoop-availability/) [Accessed, 2013].
17. Loughran S. Apache Hadoop Wiki: Jobtracker. (Available from: http://wiki.apache.org/hadoop/JobTracker/) [Accessed, 2011].
18. technologies M. The mapr distribution for apache hadoop. *White paper, MapR technologies co.*, 2011.
19. Kuromatsu N, Okita M, Hagihara K. Evolving fault-tolerance in hadoop with robust auto-recovering jobtracker. *Bulletin of Networking, Computing, Systems, and Software* 2013; **2**(1):4–11.
20. Hadoop A. Apache hadoop release note. (Available from: http://hadoop.apache.org/release.html/) [Accessed, 2013].
21. Sangroya A, Serrano D, Bouchenak S. Mrbs: towards dependability benchmarking for hadoop mapreduce. *Euro-Par 2012: Parallel Processing Workshops*, Rhodes Island, Greece, 2013; 3–12.
22. Sahoo R, Squillante M, Sivasubramaniam A, Zhang Y. Failure data analysis of a large-scale heterogeneous server environment. *International Conference on Dependable Systems and Networks*, Florence, Italy, 2004; 772–781.
23. Schroeder B, Gibson G. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing* 2010; **7**(4):337–351.
24. Schroeder B, Gibson G. The computer failure data repository (cfdr). (Available from: http://cfdr.usenix.org/) [Accessed Octorbor 6, 2012].
25. Rausand M, Hoyland A. *System Reliability Theory; Models, Statistical Methods and Applications*. John Wiley & Sons, Inc: Hoboken, NJ, USA, 2004.
26. Cvetkovic AM, Djordjevic GT, Stefanovic MC. Performance analysis of dual switched diversity over correlated weibull fading channels with co-channel interference. *International Journal of Communication Systems* 2011; **24**(9):1183–1195.
27. Schroeder B, Gibson G. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you. *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, USA, 2007; 1–16.
28. lanl lab. Operational data to support and enable computer science research. (Available from: http://institutes.lanl.gov/data/fdata/) [Accessed Octobor 6, 2012].
29. ReliaSoft. Probability plotting. (Available from: http://weibull.com/hotwire/issue8/relbasics8.htm/) [Accessed Octobor 6, 2012].
30. Wang G, Butt AR, Pandey P, Gupta K. Using realistic simulation for performance analysis of mapreduce setups. *Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance*, Garching near Munich, Germany, 2009; 19–26.
31. Wang G, Butt AR, Pandey P, Gupta K. A simulation approach to evaluating design decisions in mapreduce setups. *IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS' '09)*, Imperial College London, UK, 2009; 1–11.
32. Wang G, Butt AR, Monti H, Gupta K. Towards synthesizing realistic workload traces for studying the hadoop ecosystem. *IEEE 19th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, Raffles Hotel, Singapore, 2011; 400–408.
33. Tian F, Chen K. Towards optimal resource provisioning for running mapreduce programs in public clouds. *IEEE International Conference on Cloud Computing (CloudCom)*, Washington, DC, USA, 2011; 155–162.
34. Rizvandi NB, Taheri J, Moraveji R, Zomaya AY. A study on using uncertain time series matching algorithms for MapReduce applications. *Concurrency and Computation: Practice and Experience* 2013; **25**(12):1699–1718. DOI: 10.1002/cpe.2895.
35. Yang H, Luan Z, Li W, Qian D. Mapreduce workload modeling with statistical approach. *Journal of Grid Computing* 2012; **10**(2):279–310.