Deriving Job Completion Reliability and Job Energy Consumption for a General MapReduce Infrastructure from Single-Job Perspective

Jia-Chun Lin[*1], Fang-Yie Leu[#2], Ming-Chang Lee[*3], Ying-ping Chen[*4]
[*]Department of Computer Science, National Chiao Tung University, Taiwan
[1]kellylin1219@gmail.com, [3]mingchang1109@gmail.com, [4]ypchen@cs.nctu.edu.tw
[#]Department of Computer Science, TungHai University, Taiwan
[2]leufy@thu.edu.tw

Abstract—**MapReduce as a master-slave infrastructure consists of two master-side severs and a large number of slave-side working nodes. In this paper, we derive a job completion reliability (JCR for short) model from a single-job perspective for a general MapReduce infrastructure in which no redundancy scheme is adopted on the master side, and a cold-standby scheme is employed on the slave side. Without loss of generality, the JCR model is derived based on a Poisson distribution. In addition, we calculate the corresponding job energy consumption (JEC for short). Through the simulation and analytical results, MapReduce managers and service providers can comprehend how this infrastructure behaves and how to improve the infrastructure so as to achieve a more reliable and energy-efficient MapReduce environment.**

Keywords-MapReduce; master-slave infrastructure; job completion reliability; job energy consumption; single-job perspective; Poisson distribution

I. INTRODUCTION

MapReduce [1], a distributed programming framework, has been widely employed by many organizations/institutes, such as Apache, Yahoo, Facebook, etc., to solve their massive data-processing problems. MapReduce as a master-slave infrastructure comprises two master-side servers, called JobTracker and NameNode in Hadoop [2], and a lot of slave-side nodes, called slaves or workers. JobTracker coordinates the execution of jobs, while NameNode manages the distributed filesystem namespace of the infrastructure. Upon receiving a job *J* submitted by a user, JobTracker requests a set of slaves to execute *J*'s tasks in parallel so as to speed up the execution of *J*.

Node failures are inevitable in a large-scale computing environment, such as a cloud system [3]. Google has experienced the failure of 5 workers per MapReduce job in average [4]. The study in [5] showed that the probability of node failure rises when the scale of a system increases. To prevent the execution of jobs from being interrupted by node failures, Hadoop [2], one of the most popular open-source MapReduce implementations, utilized a cold-standby redundancy scheme on its slave side, i.e., when a slave fails, the tasks running on it cannot be finished. This scheme re-performs the task on a cold-standby node. But Hadoop by default does not provide redundant schemes for JobTracker and NameNode. This type of infrastructure has been adopted worldwide, and hence in this paper, we call it a general MapReduce infrastructure.

To our best knowledge, the job completion reliability (JCR for short) and job energy consumption (JEC for short) of a general MapReduce infrastructure have not been studied where JCR is defined as the probability that the infrastructure can complete a job, and JEC is defined as the energy consumed by the infrastructure to finish the job. To achieve a more reliable and energy-efficient computing environment, it is required to know how this infrastructure impacts its JCR and JEC. Therefore, in this study, we analyze the JCR of this infrastructure, and calculate the corresponding JEC. Without loss of generality, the JCR model is derived based on a Poisson distribution, i.e., node failure rates remain constant during the lifetime of the infrastructure. The simulation and analytical results show that this infrastructure is energy-efficient, but its master-side JCR is low, particularly when long-term jobs are submitted. It is necessary to utilize master-side redundant schemes to enhance the overall JCR.

The key contributions of this study are as follows. (1) We analyze JCR and JEC for the most widely-adopted MapReduce infrastructure, i.e., the general MapReduce infrastructure. MapReduce managers can then comprehend how this infrastructure affects its JCR and JEC. (2) Our simulation results can help MapReduce managers to determine an appropriate number of cold-standby nodes based on their resource limitations and requirements. (3) Redundant scheme designers can refer to our analytical results to propose a more reliable and energy-efficient MapReduce infrastructure.

The rest of this paper is organized as follows. Section 2 describes the background and related work of this study. Sections 3 and 4 introduce how to analyze the JCR and JEC, respectively. The simulation results are presented in Section 5. Section 6 concludes this paper and outlines our future studies.

II. BACKGROUND AND RELATED WORK

In this section, we briefly describe a MapReduce job execution flow of a MapReduce infrastructure, and the related work of this study.

A. MapReduce Job Execution Flow

Fig. 1 shows the execution flow of a job *J*. A user *U* requests slave locations from NameNode to store *J*'s job resources in step 1. After that, *U* submits *J* to JobTracker in step 2. In step 3, JobTracker initiates *J* and assigns the map (reduce) tasks of *J* to available slaves, called mappers (reducers). Before running the assigned task, a mapper or a reducer needs to retrieve *J*'s job resources from the distributed filesystem by consulting NameNode. When a mapper finishes its task, it replies to JobTracker with the disk location where the generated result resides. When all map tasks of *J* are completed, a reducer can starts running its assigned reduce task. After all reducers finish their tasks, JobTracker informs *U* of the completion of *J*.

The above flow shows that if JobTracker or NameNode fails, *J* cannot be initiated, performed, and completed. In other words, both the two servers must operate normally during the execution of *J*. Besides, each mapper needs to work properly during the execution of its assigned map task of *J*. Otherwise, this map task cannot be completed, and *J*'s reduce tasks cannot start. Also, each reducer needs to be operational from the moment when it receives the assigned reduce task of *J* to the moment when it completes the task. Otherwise, this task, also *J*, cannot be finished.

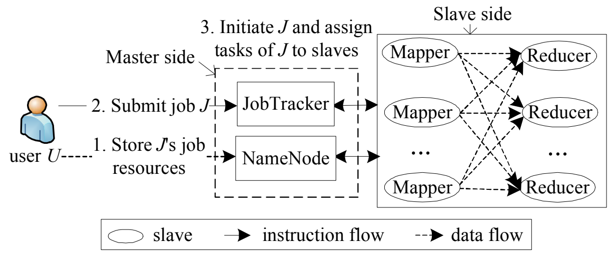


Fig 1. The execution flow of a MapReduce job *J* in a MapReduce infrastructure.

B. Related Work

Mohammad et al. [6] evaluated the reliability of a phased-mission system with a *k*-out-of-*n* load-sharing redundant scheme. Levitin et al. [7] estimated the task execution time and reliability of a multi-processing-unit hardware system. However, the two systems were not organized as a master-slave infrastructure, and the redundant schemes they utilized were not cold-standby schemes. Hence, their models cannot be directly applied to develop the JCR model of a general MapReduce infrastructure.

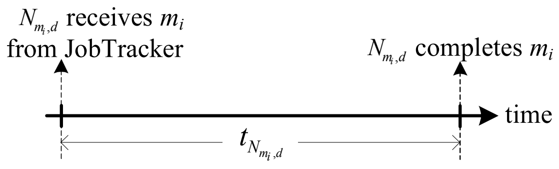
Leu et al. [8] introduced a multi-stage fault-tolerant platform to enhance the reliability of their grid-based intrusion detection system. Zhang et al. [9] estimated the downtime and availability of a system employing an active-standby redundant mechanism for both of its internal and external systems. But this model focused on system availability, rather than job completion reliability. Dai et al. [10] presented a hierarchical reliability model for grid services, and evaluated the probability that a program could be completed by a grid system. A similar model proposed by [11] studied the reliability of a program invoked by a cloud system. Other reliability models developed for software, hardware, distributed systems, wireless sensor networks, and data storage systems can be found in [12][13][14][15]. Since the characteristics of these systems are dissimilar with those of a general MapReduce infrastructure, their models are unable to be applied to this study directly.

III. Job Completion Reliability (JCR)

Assume that J is divided into x map tasks m_1, m_2, …, m_x and y reduce tasks r_1, r_2, …, r_y, where $x, y \geq 1$. Let $N_{m_i,0}$ and $N_{r_j,0}$ be the slave node that JobTracker initially assigns to perform m_i and r_j, respectively, where $1 \leq i \leq x$ and $1 \leq j \leq y$. Let the cold-standby scheme provides C cold-standby slaves, denoted by $N_{m_i} = \{N_{m_i,1}, N_{m_i,2}, \ldots, N_{m_i,C}\}$, for m_i, and C cold-standby slaves, denoted by $N_{r_j} = \{N_{r_j,1}, N_{r_j,2}, \ldots, N_{r_j,C}\}$, for r_j, $C \geq 1$. Let $N^*_{m_i}$ and $N^*_{r_j}$ be the slave nodes prepared for executing m_i and r_j, respectively. Then $N^*_{m_i} = \{N_{m_i,0}\} \cup N_{m_i}$ and $N^*_{r_j} = \{N_{r_j,0}\} \cup N_{r_j}$. Assume that a slave can perform at most one of *J*'s tasks during the execution of *J*, indicating that the maximum number of slaves running *J* is $(x + y) \cdot (C + 1)$.

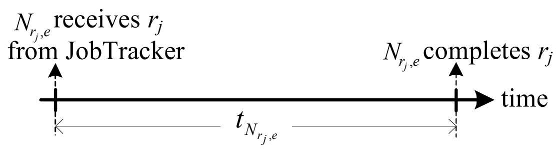
Since the cold-standby scheme invoked on the slave side may affect the length of the time period in which the master servers should be operational to finish *J*, in the following we derive the slave-side JCR first and then the master-side JCR under the assumption that all master-side servers and slave-side nodes are homogenous with the same failure rate λ following a Poisson distribution, i.e., λ remains constant during the lifetime of the infrastructure. Only node failure is considered. Other faults, such as network failures, are not addressed.

When a node $N_{m_i,d}$ receives the assignment of m_i from JobTracker, if it completely executes m_i and generates the intermediate results I_{m_i}, then the time period in which $N_{m_i,d}$ has to be available, denoted by $t_{N_{m_i,d}}$, is illustrated in Fig. 2, $0 \leq d \leq C$. Similarly, on receiving the assignment of r_j from JobTracker, if $N_{r_j,e}$ can obtain the required intermediate results to execute and complete r_j, then the time period in which $N_{r_j,e}$ has to work properly, denoted by $t_{N_{r_j,e}}$, is shown in Fig. 3, $0 \leq e \leq C$.



$N_{m_i,d}$ receives m_i from JobTracker
$N_{m_i,d}$ completes m_i
time
$t_{N_{m_i,d}}$

Fig. 2. The time period $t_{N_{m_i,d}}$ in which $N_{m_i,d}$ has to be available to complete m_i, $1 \leq i \leq x$ and $0 \leq d \leq C$.



$N_{r_j,e}$ receives r_j from JobTracker
$N_{r_j,e}$ completes r_j
time
$t_{N_{r_j,e}}$

Fig. 3. The time period $t_{N_{r_j,e}}$ in which $N_{r_j,e}$ needs to work normally to finish r_j, $1 \leq j \leq y$ and $0 \leq e \leq C$.

When the cold-standby scheme is employed on the slave side, JobTracker reassigns m_i (r_j) to $N_{m_i,1}$ ($N_{r_j,1}$) when

$N_{m_i,0}$ ($N_{r_j,0}$) crashes in $t_{N_{m_i,0}}$ ($t_{N_{r_j,0}}$), reassigns m_i (r_j) to $N_{m_i,2}$ ($N_{r_j,2}$) when $N_{m_i,1}$ ($N_{r_j,1}$) fails in $t_{N_{m_i,1}}$ ($t_{N_{r_j,1}}$), and so on. Thus, if m_i is completed by $N_{m_i,d}$, implying that $N_{m_i,d}$ works normally in $t_{N_{m_i,d}}$, but $N_{m_i,0}$, $N_{m_i,1}$, …, and $N_{m_i,d-1}$ fail in $t_{N_{m_i,0}}$, $t_{N_{m_i,1}}$, …, and $t_{N_{m_i,d-1}}$, respectively. The corresponding reliability of completing m_i by $N_{m_i,d}$, denoted by $R_{N_{m_i,d}}$, is

$$R_{N_{m_i,d}} = \prod_{a=0}^{d-1}\left(1-\exp\left(-\lambda \cdot t_{N_{m_i,a}}\right)\right)\cdot \exp\left(-\lambda \cdot t_{N_{m_i,d}}\right) \qquad (1)$$

where $\prod_{a=0}^{d-1}\left(1-\exp\left(-\lambda \cdot t_{N_{m_i,a}}\right)\right)$ is the probability that $N_{m_i,0}$, $N_{m_i,1}$, …, and $N_{m_i,d-1}$ cannot finish m_i. Let R_{m_i} be the reliability that $N^*_{m_i}$ members can complete m_i, i.e.,

$$R_{m_i} = \sum_{d=0}^{C} R_{N_{m_i,d}} = \exp\left(-\lambda \cdot t_{N_{m_i,0}}\right) + \sum\nolimits_{b=1}^{C}\left(\prod_{a=0}^{b-1}\left(1-\exp\left(-\lambda \cdot t_{N_{m_i,a}}\right)\right)\cdot \exp\left(-\lambda \cdot t_{N_{m_i,b}}\right)\right) \qquad (2)$$

Note that $R_{m_i} + \prod_{d=0}^{C}\left(1-\exp\left(-\lambda \cdot t_{N_{m_i,d}}\right)\right) = 1$ in which $\prod_{d=0}^{C}\left(1-\exp\left(-\lambda \cdot t_{N_{m_i,d}}\right)\right)$ is the probability that all members of $N^*_{m_i}$ cannot finish m_i. Let $t'_{N_{m_i,d}}$ be time period in $t_{N_{m_i,d}}$ during which $N_{m_i,d}$ is operational where $1 \le i \le x$ and $0 \le d \le C$, implying that $t'_{N_{m_i,d}} \le t_{N_{m_i,d}}$. Then in the worst case, the time period starting when JobTracker sends the assignment of m_i to $N_{m_i,0}$ and ending when m_i is completed, denoted by t_{m_i}, is

$$t_{m_i} = \sum_{f=0}^{C-1} t'_{N_{m_i,f}} + t_{N_{m_i,C}} \approx \sum_{d=0}^{C} t_{N_{m_i,d}} \qquad (3)$$

in which $t'_{N_{m_i,f}} \approx t_{N_{m_i,f}}$ means that $N_{m_i,f}$ fails when it almost completes m_i, $0 \le f \le C-1$.

Similarly, the reliability of finishing r_j by $N_{r_j,e}$, denoted by $R_{N_{r_j,e}}$, is,

$$R_{N_{r_j,e}} = \prod_{b=0}^{e-1}\left(1-\exp\left(-\lambda \cdot t_{N_{r_j,b}}\right)\right)\cdot \exp\left(-\lambda \cdot t_{N_{r_j,e}}\right) \qquad (4)$$

Let R_{r_j} be the reliability that $N^*_{r_j}$ members can complete r_j, i.e.,

$$R_{r_j} = \sum_{e=0}^{C} R_{N_{r_j,e}} = \exp\left(-\lambda \cdot t_{N_{r_j,0}}\right) + \sum\nolimits_{b=1}^{C}\left(\prod_{a=0}^{b-1}\left(1-\exp\left(-\lambda \cdot t_{N_{r_j,a}}\right)\right)\cdot \exp\left(-\lambda \cdot t_{N_{r_j,b}}\right)\right) \qquad (5)$$

Note that $R_{r_j} + \prod_{e=0}^{C}\left(1-\exp\left(-\lambda \cdot t_{N_{r_j,e}}\right)\right) = 1$ in which $\prod_{e=0}^{C}\left(1-\exp\left(-\lambda \cdot t_{N_{r_j,e}}\right)\right)$ is the probability that all members of $N^*_{r_j}$ cannot complete r_j. Let $t'_{N_{r_j,e}}$ be a time period in $t_{N_{r_j,e}}$ during which $N_{r_j,e}$ is operational where $1 \le j \le y$ and $0 \le e \le C$, implying that $t'_{N_{r_j,e}} \le t_{N_{r_j,e}}$. Hence, in the worst case, the time period starting when JobTracker sends the assignment of r_j to $N_{r_j,0}$ and ending when r_j is completed, denoted by t_{r_j}, is

$$t_{r_j} = \sum_{g=0}^{C-1} t'_{N_{r_j,g}} + t_{N_{r_j,C}} \approx \sum_{e=0}^{C} t_{N_{r_j,e}} \qquad (6)$$

where $t'_{N_{r_j,g}} \approx t_{N_{r_j,g}}$ represents that $N_{r_j,g}$ fails when it almost finishes r_j, $0 \le g \le C-1$.

Consequently, the reliability that the slave side with $x+y$ initially-assigned slaves and $C \cdot (x+y)$ cold-standby slaves can finish *J*'s all map and reduce tasks, denoted by R^J_{SS}, is

$$R^J_{SS} = \prod_{i=1}^{x} R_{m_i} \cdot \prod_{j=1}^{y} R_{r_j} \qquad (7)$$

Let the execution time of *J* be a time period starting from the moment when JobTracker receives the submission of *J* from a user *U* to the moment when *J*'s reduce tasks are all finished. During this execution time, JobTracker and NameNode must operate normally to make sure all tasks of *J* can be successfully assigned and performed. Otherwise, *J* cannot be completed. Assume that on receiving *J*, JobTracker can immediately and simultaneously assign and send *J*'s map tasks to slave nodes, and finish the assignment in a very short period of time, e.g., several milliseconds. Also, when all map tasks of *J* are completed, JobTracker can immediately and simultaneously assign *J*'s reduce tasks to slave nodes. We further assume that a task assignment can be instantly delivered from JobTracker to a slave, implying that in the worst case the execution time of *J*, denoted by t_J, is

$$t_J = \max\left(t_{m_1}, t_{m_2}, \dots, t_{m_x}\right) + \max\left(t_{r_1}, t_{r_2}, \dots, t_{r_y}\right) \qquad (8)$$

Let $R^J_{JobTracker}$ ($R^J_{NameNode}$) be the reliability that JobTracker (NameNode) operates normally during the execution of *J*, i.e., the probability that JobTracker (NameNode) works normally in t_J. Then,

$$R^J_{JobTracker} = R^J_{NameNode} = \exp\left(-\lambda \cdot t_J\right) \qquad (9)$$

Hence, the JCR that the slave side and the master side can finish *J*'s all map and reduce tasks, denoted by R^J, is

$$R^J = R^J_{SS} \cdot R^J_{JobTracker} \cdot R^J_{NameNode} \qquad (10)$$

IV. Job Energy Consumption (JEC)

Let C^J be the energy that a general MapReduce infrastructure consumes to finish *J*. Assume that all slaves' (master servers') power consumption rates, denoted by δ_{SS} (δ_{MS}), are the same. Hence,

$$C^J = \delta_{SS} \cdot T^J_{SS} + \delta_{MS} \cdot T^J_{MS} \qquad (11)$$

where T^J_{SS} (T^J_{MS}) is the cumulative time consumed by the slave-side nodes (master servers) during the execution of *J*. If N_{m_i,d_i} is the member of $N^*_{m_i}$ that finishes m_i, and N_{r_j,e_j} is the member of $N^*_{r_j}$ that completes r_j, $0 \le d_i, e_j \le C$, then

(1) $0 \le t'_{N_{m_i,a}} < t_{N_{m_i,a}}$ and $0 \le t'_{N_{r_j,b}} < t_{N_{r_j,b}}$ where $a = 0,1,\dots,d_i - 1$, and $b = 0,1,\dots,e_j - 1$;

(2) $t'_{N_{m_i,d_i}} = t_{N_{m_i,d_i}}$ (see Fig. 2) and $t'_{N_{r_j,e_j}} = t_{N_{r_j,e_j}}$ (see Fig. 3);

(3) $t'_{N_{m_i,d_i+1}} = t'_{N_{m_i,d_i+2}} = \cdots = t'_{N_{m_i,C}} = 0$ and $t'_{N_{r_j,e_j+1}} = t'_{N_{r_j,e_j+2}} = \cdots = t'_{N_{r_j,C}} = 0$. The reason is that when $N_{m_i,0} \sim N_{m_i,d_i}$ ($N_{r_j,0} \sim N_{r_j,e_j}$) perform m_i (r_j), $N_{m_i,d_i+1} \sim N_{m_i,C}$ ($N_{r_j,e_j+1} \sim N_{r_j,C}$) staying in their cold-standby modes are not operational and of course do not consume time to execute m_i (r_j).

Therefore, T^J_{SS} is

$$T^J_{SS} = \sum_{i=1}^{x}\left(\sum_{d=0}^{C} t'_{N_{m_i,d}}\right) + \sum_{j=1}^{y}\left(\sum_{e=0}^{C} t'_{N_{r_j,e}}\right) = \sum_{i=1}^{x}\left(\sum_{a=0}^{d_i-1} t'_{N_{m_i,a}} + t_{N_{m_i,d_i}}\right) + \sum_{j=1}^{y}\left(\sum_{b=0}^{e_j-1} t'_{N_{r_j,b}} + t_{N_{r_j,e_j}}\right) \qquad (12)$$

In the worst case,

$$T^J_{SS} = \sum_{i=1}^{x}\left(\sum_{d=0}^{C} t_{N_{m_i,d}}\right) + \sum_{j=1}^{y}\left(\sum_{e=0}^{C} t_{N_{r_j,e}}\right) \qquad (13)$$

On the other hand, the cumulative time consumed by the master side to finish *J*, i.e., T^J_{MS}, comprises the times that JobTracker and NameNode spend during the execution of *J*. That is,

$$T^J_{MS} = 2t_J \qquad (14)$$

Hence, based on Eqs. (11), (13), and (14), in the worst case, C^J is

$$C^J = \delta_{MS} \cdot 2t_J + \delta_{SS} \cdot \left(\sum_{i=1}^{x}\left(\sum_{d=0}^{C} t_{N_{m_i,d}}\right) + \sum_{j=1}^{y}\left(\sum_{e=0}^{C} t_{N_{r_j,e}}\right)\right) \qquad (15)$$

V. Simulation and Comparsion

We simulated job execution in a general MapReduce infrastructure and analyzed the corresponding JCRs and JECs. Let $\lambda = 0.0001$ per hour for all master-side and slave-side nodes. Let $\delta_{SS} = 0.3$ kW, and $\delta_{MS} = 0.5$ kW. To further show how different numbers of cold-standby nodes influence the JCR and JEC, this infrastructure was tested on five settings of C, including $<C = v>$, $v = 1,2,\dots,5$. Fifteen jobs with different lengths of task execution time, i.e., $t_{N_{r_j,e}}$, were simulated (see Table 1), and each of them was divided into 256 map tasks (i.e., $x = 256$) and 128 reduce task (i.e., $y = 128$). To reduce the simulation complexity, we further assumed that $t_{N_{m_i,d}}$ of a job is equal to $t_{N_{r_j,e}}$ of the job, $1 \le i \le 256$, $1 \le j \le 128$, and $0 \le d, e \le C$.

TABLE I. The fifteen jobs tested in this study. Note that the task execution time represents $t_{N_{r_j,e}}$ (also $t_{N_{m_i,d}}$)

Job No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Task execution time (hour)	0.125	0.25	0.5	1	2	4	8	16	32	64	128	256	512	1024	2048
$\boldsymbol{x}$	256														
$\boldsymbol{y}$	128														

TABLE II. The number of times that a job was finished in its thirty submissions

Job No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\boldsymbol{C = 1}$	30	30	30	30	30	30	30	30	30	29	27	27	19	12	2
$\boldsymbol{C = 2}$	30	30	30	30	30	30	30	30	30	29	29	26	20	11	2
$\boldsymbol{C = 3}$	30	30	30	30	30	30	30	30	30	27	28	26	21	12	3
$\boldsymbol{C = 4}$	30	30	30	30	30	30	30	30	30	28	29	25	21	12	4
$\boldsymbol{C = 5}$	30	30	30	30	30	30	30	30	30	28	28	27	22	11	4

TABLE III. Average job execution times of the fifteen tested jobs (Unit: hour)

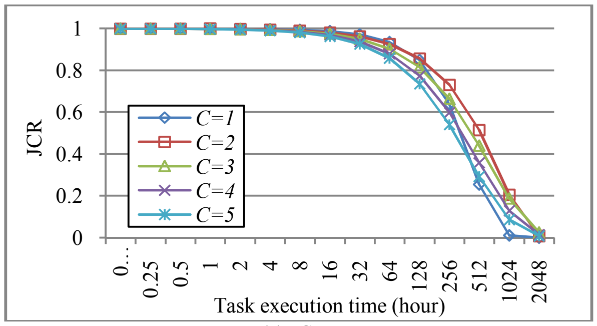
Job No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\boldsymbol{C = 1}$	0.25	0.5	1.0	2.0	4.0	8.2	17.9	35.2	79.2	177.1	427.7	900.9	1932.6	3974.6	8059.1
$\boldsymbol{C = 2}$	0.25	0.5	1.0	2.0	4.1	8.3	18.1	38.5	81.9	171.0	433.5	898.5	2005.1	4478.0	10232.8
$\boldsymbol{C = 3}$	0.25	0.5	1.0	2.0	4.1	8.2	18.9	36.7	75.3	177.8	409.8	911.6	1983.9	4463.1	10923.9
$\boldsymbol{C = 4}$	0.25	0.5	1.0	2.0	4.0	8.4	18.2	35.8	78.7	179.4	424.2	933.9	1997.5	4486.5	10772.1
$\boldsymbol{C = 5}$	0.25	0.5	1.0	2.0	4.1	8.5	18.2	37.0	76.5	180.1	422.5	931.1	1988.5	4595.7	11346.5

Each job *J* was submitted thirty times, in which the number of times that *J* was finished is listed in Table II. On $C = 1$, jobs 1~9 were all finished in their thirty-time submissions, implying that $C = 1$ (i.e., one cold-standby slave for each task) was sufficient for each of these 9 jobs. But this was not true for jobs 10~15 since their tasks were longer so that the probability that a slave could not finish its assigned task was higher. Table II also shows that increasing C still cannot effectively enhance the probability of completing jobs 10~15. This is because the master side had low JCRs on these six jobs, even though their slave sides could finish the assigned tasks.

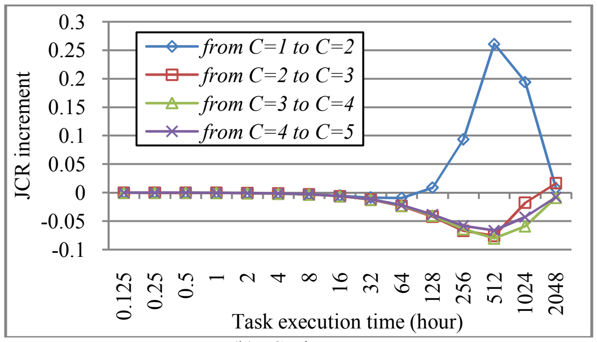
Table III lists the average execution times of these jobs. When C was higher, the execution time of a job (excluding job 15) did not monotonously increase, and the differences of the execution times of a short-term job (e.g., one of jobs 1~7) on different Cs were insignificant. This is because most tasks of a short-term job were completed by their initially-assigned slaves, i.e., $N_{m_i,0}$ and $N_{r_j,0}$, $1 \le i \le 256$, $1 \le j \le 128$. But when a long-term job (e.g., one of jobs

8~15) was tested, most tasks of the job needed to be reexecuted several times, e.g., k times, by their first k cold-standby slaves. Hence, its execution-time differences on different Cs were evident.

Figs. 4a and 4b respectively illustrate the JCRs and JCR increments of the general MapReduce infrastructure. The JCRs shown in Fig. 4a decreased slowly when task execution time extended from 0.125 hour to 16 hours. But when task execution time further increased, due to the decrease of slave-side and master-side JCRs, the overall JCRs declined sharply. When task execution time is between 32 hours and 1024 hours, the JCRs on $C = 2$ were higher than those on $C = 1$ (see Fig. 4a), implying that increasing C from 1 to 2 can dramatically raise the JCR. This phenomenon can also be observed in Fig. 4b. But when C was further raised, the JCR increments were reduced. Fig. 4b shows that increasing C to 3, 4, and 5 did not bring any JCR increment since the master-side did not employ redundant schemes, and hence the overall JCR was not further enhanced. When task execution time was 2048 hours, the JCRs on all Cs were low. The reason is the same.



(a) JCRs

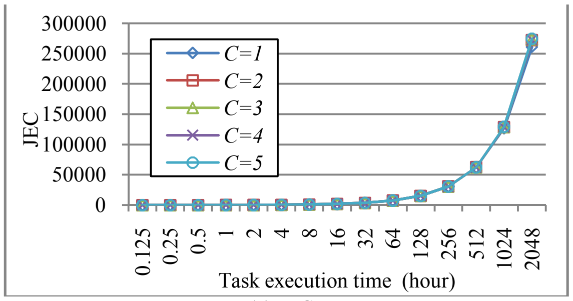


(b) JCR increments

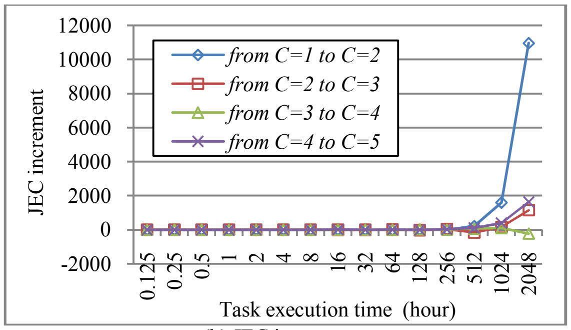
Fig. 4. The JCRs and JCR increments of the MapReduce infrastructure on five settings of C.

Figs. 5a and 5b illustrate the average JECs and JEC increments, respectively. When task execution time increased doubly on a specific C, the corresponding JECs raised almost proportionally. Fig. 5a also shows that when task execution time was shorter than 512 hours, the five JEC curves were almost overlapped, implying that increasing C did not notably boost the JECs. Fig. 5b also reflects this. The key reason is that most tasks of these jobs (i.e., jobs 1~13) were finished by their initially-assigned slaves or their first cold-standby slaves. Other cold-standby slaves did not consume time to perform these tasks.

When task execution time further grew to 2048 hours (i.e., job 15 was submitted), increasing C from 1 to 2 resulted in an evident JEC increment (see Fig. 5b), but further increasing C to 3, 4, or 5 caused a decreasing JEC increment. The key reason is that most tasks of this job were completed by their second cold-standby slaves, and some were completed by their third cold-standby slaves. Only one or two were finished by their first cold-standby slaves. Hence, if the job is completed on $C = 1$, the corresponding JEC will be much lower than the JEC on $C = 2$.



(a) JECs



(b) JEC increments

Fig. 5. The average JECs and JEC increments of the MapReduce infrastructure on five settings of C.

VI. Conclusions and Future Work

In this study, we have derived JCR and JEC for a general MapReduce infrastructure. We also presented and discussed our simulation results on fifteen jobs and different numbers of cold-standby slaves. The results show that the slave side of the general MapReduce infrastructure was reliable, but the master side of the infrastructure was not. Hence, establishing a master-side redundant scheme for this infrastructure is required, especially when users often submit long-term jobs to this infrastructure. In addition, due to adopting the cold-standby scheme on the slave side, this infrastructure was very energy-efficient since increasing the number of cold-standby slaves for each task did not considerably raise the JEC. In fact, from our simulation results, MapReduce managers can comprehend the impact of the general MapReduce infrastructure on its JCR and JEC, and the shortcomings of this infrastructure.

In the future, we would like to extend the analyses of JCR and JEC from a multi-job perspective by considering the queueing delays of the master side and slave side. We also plan to further analyze the JCRs and JECs of other MapReduce infrastructures that employ different redundant schemes, and compare their performances.

Acknowledgment

We would like to thank the anonymous reviewers for their helpful comments. The work was supported in part by the GREENs project of TungHai University and the National Science Council, Taiwan under Grants NSC 101-2221-E-009-003-MY3 and NSC 101-2628-E-009 -024 -MY3.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communication of the ACM*, Vol. 51, Issue 1, pp. 107–113, 2008.

[2] Apache Hadoop, http://hadoop.apache.org (16.09.12)

[3] S. Y. Ko, I. Hoque, b. Cho, and I. Gupta, "Making Cloud Intermediate Data Fault-Tolerant," *Proc. of the 1st ACM symposium on Cloud computing, 2010*, pp. 181–192.

[4] J. Dean, "Experiences with MapReduce, an Abstraction for Large-Scale Computation," In *Keynote I: PACT*, 2006.

[5] B. Schroeder and G. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," *IEEE Transactions on Dependable and Secure Computing*, Vol. 7, Issue 4, pp. 337–350, 2010.

[6] R. Mohamman, A. Kalam, and S.V. Amari, "Reliability Evaluation of Phased-Mission Systems with Load-Sharing Components," *Proc. of the Annual Reliability and Maintainability Symposium (RAMS)*, Jan. 2012, pp. 1–6.

[7] G. Levitin, M. Xie, and T.L. Zhang, "Reliability of Fault-tolerant Systems with Parallel Task Processing," *European Journal of Operational Research*, Vol. 177, Issue 1, pp. 420–430, 2007.

[8] F.Y. Leu, C.T. Yang, F.C. Jiang, "Improving Reliability of a Heterogeneous Grid-based Intrusion Detection Platform using Levels of Redundancies," *Future Generation Computer Systems*, Vol. 26, Issue 4, pp. 554–568, 2010.

[9] X. Zhang, H. Pham, and C. R. Johnson, "Reliability Models for Systems with Internal and External Redundancy," *International Journal of Systems Assurance Engineering and Management*, Vol. 1, No. 4, pp. 362–369, 2010.

[10] Y.-S. Dai, Y. Pan, and X. Zou, "A Hierarchical Modeling and Analysis for Grid Service Reliability," *IEEE Transactions on Computers*, Vol. 56, Issue 5, pp.681–691, 2007.

[11] Y.-S. Dai, B. Yang, J. Dongarra, and G. Zhang, "Cloud Service Reliability: Modeling and Analysis," *15th IEEE Pacific Rim International Symposium on Dependable Computing*. 2009.

[12] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, "Hadoop High Availability through Metadata Replication," *Proc. of the first international workshop on Cloud data management*, New York, NY, USA: ACM, 2009, pp. 37–44.

[13] Y. Xiang, T. Chantem, R. P. Dick, X.S. Hu, and L. Shang, "System-Level Reliability Modeling for MPSoCs," *Proc. of the eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2010, pp. 297–306.

[14] K. M. Greenan, J. S. Plank, and J. J. Wylie, "Mean time to meaningless: MTTDL, Markov models, and storage system reliability," *Proc. of the USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2010, pp. 1–5.

[15] V. Venkatesan and I. Iliadis, "A General Reliability Model for Data Storage Systems," IBM Research - Zurich, Tech. Rep. RZ 3817, 2012.