# AVAILABILITY MODELING AND ASSURANCE OF MAP-REDUCE COMPUTING

By

ZUQIANG KE

Bachelor of Engineering in Computer Science and Technology

Shanghai Jiao Tong University

Shanghai, China

1987

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2017

AVAILABILITY MODELING AND ASSURANCE

OF MAP-REDUCE COMPUTING

Thesis Approved:

Dr. Nohpill Park

Thesis Adviser

Dr. Eric Chan-Tin

Dr. David Cline

ACKNOWLEDGEMENTS

It would not have been possible to write this thesis without the help and support of the people around me, and I only can mention some of them here.

Primarily, to my advisor Dr. Nohpill Park for supporting and encouraging me during my graduate studies here, particularly to him for allowing me to ask questions and challenging me to go beyond myself. His guidance helped me to overcome many difficulties during this thesis research. I am grateful to him for keeping trust in my abilities until the completion of the thesis.

Next, I would like to thank Dr. Eric Chan-Tin and Dr. David Cline for kindly agreeing to serve on my thesis committee. I sincerely acknowledge them for providing insightful comments on my work.

I would not have come this far without the unconditional love, support and sacrifices of my father Jintai Ke, my mother Yingzi Zheng. In spite of being thousands of miles away, they gave me the strength to see through the completion this thesis.

Finally, this thesis is dedicated to my wife, Hongyu Wang, and to my daughter, Yiling Ke for always offering their love and support.

Name: ZUQIANG KE

Date of Degree: DECEMBER, 2017

Title of Study: AVAILIBILITY MODELING AND ASSURANCE OF MAP-REDUCE COMPUTING

Major Field: COMPUTER SCIENCE

Abstract: This thesis proposes a new analytical model to evaluate the availability of Map-Reduce computing on a Hadoop platform. Map-Reduce computing is represented by a queueing model in order to trace flow of Map-Reduce jobs of their arrivals and departures in the course of computation. The objective of this analytical model is to evaluate the probability for a Map-Reduce computation to be available at an instant of time, referred to as availability. The set of variables taken into account in this model lists the number of Map-Reduce jobs, the number of servers (or referred to as the worker nodes in this thesis) engaged, along with a few constants such as job arrival/completion rates and the worker node failure/repair rates. The proposed model provides a comprehensive yet fundamental basis to assure and ultimately optimize the design of Map-Reduce computing in terms of availability with reference to its performance in a simultaneous manner. Parametric simulations have been conducted and demonstrated efficacy of the proposed model in assessing the availability and the cost for achieving the availability with respect to throughput as well as turnaround time.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

# CHAPTER I

## INTRODUCTION

It is expected to be extensively exercised to lease a space and its service on a cloud, today and in the near future. Cloud system is built of millions of inexpensive disks in the form of data centers [15]. Data centers are the primary resources and cost to facilitate massive computations, namely big data computing; and thus how to maximize the efficiency or in other words how to minimize the amount of engaged resources is the key to the success of big data computing. Map-reduce computing is a type of big data computing and will be the primary computational model in this work.

In fact, the effectiveness and efficiency of big data computing is determined by the availability of the nodes in the cloud. As the clouds and the data centers are primarily built of inexpensive disk racks, and further as the higher availability is realized the less amount of resources required for computation, ultimately optimizing the cost for data centers can be done by minimizing the amount of computational resources. Therefore, availability will ultimately determine the quality of cloud service by minimizing downtime of service, especially in mission and safety critical sectors.

In this context, availability of Map-Reduce computing is the primary interest and concern. The specific research problem to be addressed and resolved in this thesis is how

to model and assure the availability of Map-Reduce computing in a theoretical manner yet with an extensive and practical set of variables and constants.

There has been few adequate yet practical availability model found to the best of our knowledge. The best and citable research work is the queueing-based model as presented in [1]. In [1], it has presented their proposed model to evaluate the performance of the system by using two variables, one for the number of requests (or jobs) in the queue and the second variable determines which queue the request(s) is (are) being handled by either LB (Load Balancer) or VMs (Virtual Machines) during the computation. Evidently, Map-Reduce job workload and the worker node availability are not the interests to the authors of [1], however, the model demonstrated an efficacy of queueing model to capture the behavior of the typical cloud application.

The proposed availability model in this thesis leverages queueing theory [19] and Discrete Time Markov Chain to define the impact factors of various distributed and parallel architectures in a unified specification, and uses the probabilistic analysis and statistics to analyze the complexity of Map-Reduce computing. This model introduces a new tuple of variables such as number of Map-Reduce jobs and number of servers (or the worker nodes) engaged (or failed) in the computation, which demonstrates an effective yet practical capability to model and assure the behavior of the computation specifically in the availability context while it can still provide the capability to assess the performance of the system along the way. Note that our proposed model can be solved in $O(n^3)$, where $n$ is either the number of Map-Reduce jobs or number of the worker nodes. Also, note that the increase of the order of model from $O(n^2)$ as in [1] is inevitable due to an extra variable incorporated. However, this overhead is the achievable

minimum in order to solve the availability problem. We also provide the mathematical functions to predict turnaround time for jobs and number of jobs in the platform as well as the job departure rate (mean of throughput). Parametric simulations have been conducted in order to observe the trends of the availability and the impact of the availability on the performance, particularly throughput and turnaround time, based on the proposed analytical model and the impact of the number of the worker nodes in Map-Reduce computing.

# CHAPTER II

## PRELIMINARIES AND REVIEW

The framework of Map-Reduce performs task scheduling and monitoring and re-executing failed tasks [20]. It consists of $n$ worker nodes (or slaves), each of which operates on a number of CPUs and disks [20], as illustrated 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) [17]. Each worker node runs two daemons: (1) the Task Tracker 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 [17]. The worker node is a shared resource, which supports large number of tasks to share the computing resource concurrently.

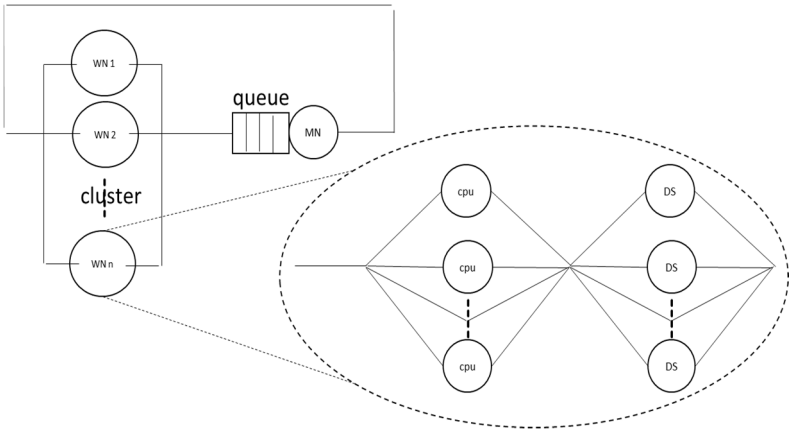


Figure 1 Map-Reduce Framework: [20]

A Map-Reduce job consists of a group of Map and Reduce tasks performing some data-intensive computation [20]. The Map task executes a user-defined map function for each key/value pair in its input [20]. The Reduce task is executed in a shuffle, sort and a reduce phase [20]. 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 [20]. The procedure is illustrated Figure 2 as follows.

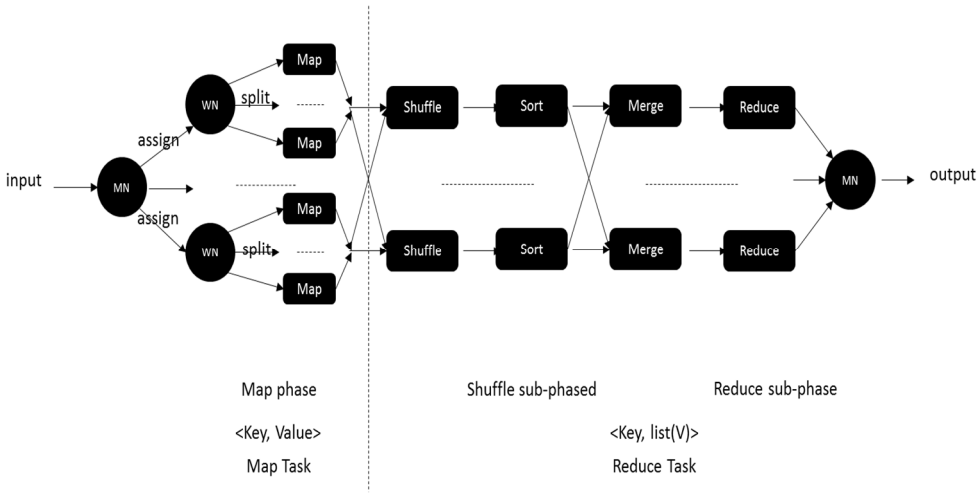


Figure 2 Map-Reduce workload: [20]

A Map-Reduce job reads one or more input files and produces one output file [20]. Each file is logically partitioned by defining the key range and is processed by each worker node [20]. Each partition can be further divided into sub-partitions of equal size, called splits. After the partition has been split, $N_m$ map tasks will be generated by a pre-defined function map(). Since a number of threads can be created and be available in the worker nodes to execute the tasks in parallel, Map-Reduce framework define the thread number $p_m$ by given the variable $N_{slot}^m$ [20]. For each split register, it executes the function map() and produce one or more output registers [20]. Map-Reduce sorts and stores output registers into a temporary file in the local file system [20].

A reduce task is responsible for processing a range of keys that have been generated by the map task [20]. One reduce task has two phases, namely, shuffle and reduce. Map-Reduce framework generates $N_{sh}$ shuffle tasks, each one responsible for processing the results of a map task, and run $p_r$ threads in parallel and transfers the data from the temporary file that have been produced by the maps that matches the key range of reduce to which the shuffle belongs. In the shuffle phase, a partial sort is performed, and the registers are written into an output file in the local file system of worker node for reduce phase [20]. As soon as all partial sorts completed, a final sort is performed, which merges the temporary files produced by the sort and shuffle [20]. Finally, $N_r$ reduce tasks will be generated by a pre-defined function reduce(), and the reduce task runs on $p_r$ threads that is given by Map-Reduce framework parameter $N_{slot}^r$ in parallel to read this merged file, and write the results into HDFS [20].

Map-Reduce has a master-slave design for its storage system [27]. In the storage system, Name node handles the metadata operations, while data node handle the read/writes

initiated by clients [27]. Files are divided into fixed-sized blocks, each stored at a different data node that is connected to the worker node. Files are read-only, but append operations may be performed in some implementations [17]. The storage systems use replication for reliability and load balancing purposes. Hadoop Distributed File System supports a configurable number of replicas per file; by default, each block of a file is replicated three times. The procedure of replica placement is shown in Figure 5 as follows.

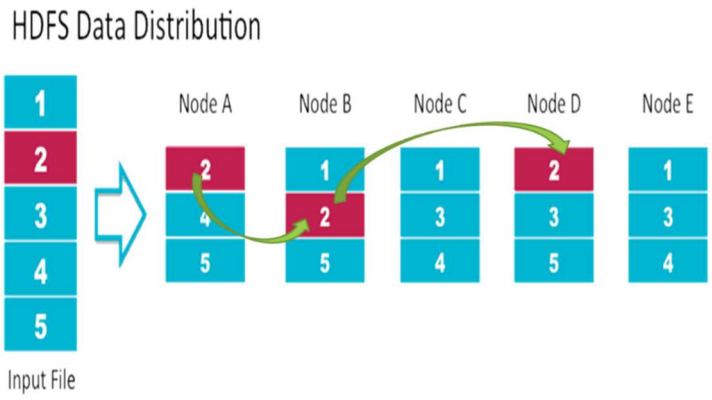


Figure 3 Hadoop Distributed File System (HDSF) data distribution

The first replica of a block goes to the node writing the data; the second, to a random node in the same rack; and the last, to another random node [27]. This design provides a good balance between being insensitive to correlated failures (e.g., whole rack failure) and minimizing the inter-rack data transmission [27].

# CHAPTER III

## PROPOSED AVAILABILITY MODEL

Queueing theory is an appropriate methodology to model the flow of the Map-Reduce computation as is to be investigated in this research [1]. Our proposed research will be centered around the availability model to be developed, and it will be a highly complex queueing model. The availability model will ultimately serve as a basis for how to guide and design the testing, diagnosis and error-tolerant execution of map-reduce computations.

The motivation of our work is that to the best of our knowledge, there is no evident justification for the replication factor to be set at 3 [2]. In other words, there is no evidence that the availability peaks out at 3 or what so ever. Either under- or over-shooting the availability (i.e., under- or over-replication of data [14]) could result in a disastrous consequence especially in healthcare-related computational systems in which they are shooting at four 9's below the decimal point for the availability (i.e., availability 0.9999 or higher), in other words about 7 minutes of downtime of the system per month in case of under-shooting; and may also result in resource waste such as triplex data centers in case of extreme over-shooting (i.e., over-replication). As such, there are various systems that require an extremely high availability such as mission, safety and deadline-critical real time systems, to mention a few. Therefore, there is an exigent need

for a solid theoretical foundation to facilitate identification of the optimal level of error/fault-tolerance [3,4,5,6] that is not supposed to be either too high to waste the resources nor too low to degrade the availability of the system. Further, the model can monitor and facilitate the testing, diagnosis and error-tolerance towards its theoretical optimality. Ultimately, the model will allow a simultaneous evaluation of the performance (e.g., turnaround time and throughput) as well as the availability, thereby making it possible to identify the theoretical break-even point between performance and availability.

A generic and naive definition of the availability is $\frac{MTTF}{MTTF+MTTR}$ [6], where *MTTF* and *MTTR* denote mean time to failure and mean time to repair, respectively, which can be as a single-variate model. However, it is seriously constrained from any extensive multi-variate analysis to take such variables as the number of map processes, reduce processes, failure rate of the name node and the rate to stay operational provided a certain number of nodes with spares into account, to mention a few. However, this model may serve for a comparison purpose as the baseline model.

In [1], a sound analytical model has been developed to evaluate a simple cloud system. The state in [1] is defined by a double-tuple, that is the number of processes and the location of the process. The proposed model incorporates an extra key variable in order to make it traceable of the availability of the system. Having the extra variable raises the complexity of the availability by an order of magnitude, from quadratic to cubic, which appears quite a manageable cost for the extra gain to be made.

There has been extensive research conducted on the performance of Map-Reduce computing in particular [7,8,9]. There have been various fault tolerance schemes proposed [10,11,12,13], however, few adequate research have been reported at the fundamental and theoretical level to address and resolve the availability issues.

The challenge to develop an efficient and reasonably accurate analytical availability model is that it must capture, with reasonable accuracy, the various factors that will influence the stability of a Map-Reduce computing. Comparing to the analytical model [1], the availability model is more complicated and cumbersome, because there are a necessity 3-tulpes random variable that is required to define a Map-Reduce state space and a very complex Map-Reduce workload. There are the following four steps as shown in the Table 1, that illustrates the development process of the proposed availability model:

Table 1: Key steps for solving proposed availability model.

| Step | Name | Actions |
|---|---|---|
| 1 | Job arrival model: | To analyze Map-Reduce job arriving process, then propose a suitable and reasonable job arrival model. |
| 2 | Job departure model | To analyze Map-Reduce job workload, then propose an approximate analytical model for Map-Reduce job service time |

| | | |
|---|---|---|
| 3 | Node repair process | To analyze Map-Reduce resilience strategy, then propose a queueing model to evaluate the node availability. |
| 4 | Map-Reduce availability model | To propose an availability model, then analyze and solve the model. |

### 3.1 Job Arrival Model

Map-Reduce job arrival model is defined for the availability model where it can be considered as a stochastic process for counting the random job arrival events. This arrival process is shown in Figure 4: the job arriving from an Ethernet is queued in the master node and then the first job is assigned to the worker node. In the worker node, the job will be split into several map tasks and then follow the shuffle and reduce tasks. After all the tasks generated by a job are completed, the job is de-queued in the master node.

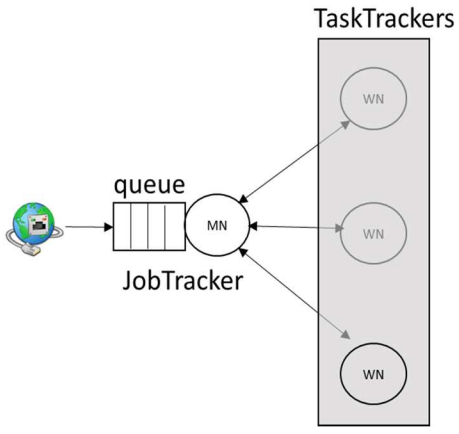


Figure 4 A finite length queue with a Master node and $n$ Worker nodes

Assuming that the model is a stationary time series model and there is no relationship between arrival jobs, Therefore, we have a constant value of the mean of the job arriving rate $\lambda = \lim_{n\to\infty} \frac{\sum x_t}{n} = \frac{\#\ of\ arrivals}{t}$.

By observing the arrival job to the master node for the time interval$(0, t]$, the time interval $(t)$ can be divided into $n$ subintervals of length $t/n$. Since the job arrival is independent, then for a sub-interval $\Delta t = t/n$, $k$ jobs arrival is like the $n$ sub-intervals as building a sequence of Bernoulli trials with the probability of success $p = \lambda t/n$. It follows the probability of $k$ job arrivals in a total of $n$ sub-interval each with a duration $t/n$, and then we have the probability as following:

$$P_{N(t)}(N(t) = k) = \binom{n}{k}(\frac{\lambda t}{n})^k(1 - \frac{\lambda t}{n})^{n-k} = \frac{n(n-1)\dots(n-k+1)}{k!\,n^k}(\lambda t)^k(1 - \frac{\lambda t}{n})^{n-k}$$

$$= \frac{n}{n}\frac{(n-1)}{n}\dots\frac{(n-k+1)}{n}\frac{(\lambda t)^k}{k!}(1 - \frac{\lambda t}{n})^{n-k}$$

According to CISCO article "Troubleshooting Ethernet Collisions" [24], if more than one messages in transmission at the same time (collision in sub-interval: $\lim_{n\to\infty} t/n$), the collision messages will be retransmitted. Therefore, $\Delta t = t/n$ should be as small as possible, in order to make sure there is at most one job arrive in the sub-interval. Therefore, have the probability as:

$$P_{N(t)}(N(t) = k) = \lim_{n\to\infty}\binom{n}{k}(\frac{\lambda t}{n})^k(1 - \frac{\lambda t}{n})^{n-k} = \frac{(\lambda t)^k}{k!}\lim_{n\to\infty}(1 - \frac{\lambda t}{n})^n$$

Since, $\lim_{h\to 0}(1 + h)^{1/h} = e$, we set $-\lambda t/n = h$, then have:

$$P(N=k)=\frac{(\lambda t)^k}{k!}\lim_{n\to\infty}\left\{\left[1-\frac{\lambda t}{n}\right]^{-\frac{n}{\lambda t}}\right\}^{-\lambda t}=\frac{(\lambda t)^k}{k!}\left[\lim_{h\to 0}(1+h)^{1/h}\right]^{-\lambda t}=\frac{(\lambda t)^k}{k!}e^{-\lambda t}$$

$$where\ k=0,1,2\ldots \qquad (3.1)$$

Apparently, it is a Poisson distribution. Therefore, Poisson process reflects the process of Map-Reduce job arrival theoretically. It is also confirmed that the job arrivals be well modeled by Poisson process in the wide-area traffic. According to the experimental report [28]: "We find that user-initiated TCP session arrivals, such as remote login and file-transfer, are well-modeled as Poisson processes with fixed hourly rates, but that other connection arrivals deviate considerably from Poisson." and the fact that Map-Reduce job arrivals are client-initiated TCP connections, Poisson process is a good model for counting the number of the job arrivals.

### 3.2 Job Departure Model

The departure model can be considered as a renewal process and can be used to predict the independent, identically distributed and nonnegative random completion time of a Map-Reduce job. The Map-Reduce job execution has following characters as the Table 2:

Table 2. Characters of Map-Reduce job execution

| ID | Character |
|---|---|
| 1 | Jobs are executed across multiple worker nodes: the map phase is partitioned into the map tasks and the shuffle and reduce phases are partitioned into the reduce tasks |

| | |
|---|---|
| 2 | The worker node automatically executes map and reduce tasks in parallel with time-sharing CPU schedule |
| 3 | The job has a different number of Map-Reduce tasks in terms of different sizes of input files |
| 4 | The data block for every Map-Reduce task has the same size |

By studying the characters of the job execution, the following problems need to be solved:

1. Since Map-Reduce tasks are executed in a time sharing and parallel model due to multi-user operation system. The probability density function of the task service time is: $f(x) = \sum_{i=1}^{n} \alpha_i \mu_i e^{-\mu_i x}$ $and$ $\sum_{i=1}^{n} \alpha_i = 1$. Apparently, it is not an exponential distribution; see [23, Page 446].

2. Since Map-Reduce job will be broken into many map tasks and then each map task can lead Map-Reduce framework to generate the reduce task, no single task departure rate can precisely represent a whole Map-Reduce job departure rate.

3. Since all the reduce tasks are made by Map-Reduce framework based on the results of the map tasks, the whole job execution flow is in a time sequence order, it is a typical PH distribution; see [25, Page 33-60].

For the first problem, let's consider the map tasks are running on a time-sharing CPU schedule as Figure 6:

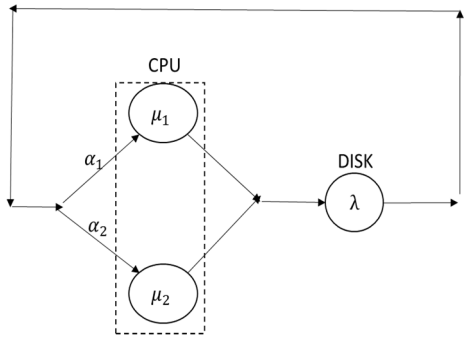


Figure 6 CPU time-sharing model

In the Figure 6, assume that the data node I/O service time is exponentially distributed with a constant rate $\lambda$ and the size of data block for splitting map task can be small enough so that the map tasks can complete within two quantum CPU times. Therefore, there are two phases hyper-exponential CPU service time, each one has the constant rate $\mu_1$ and $\mu_2$ and the same chance to gain CPU resource in each phase. Then we have the two-tuples $(i, j)$ task's state, where $i$ is number of tasks in the phase 1, and $j$ is number of tasks in the phase 2. Then the state space and state diagram are as following:

$$[(0,0), (1,0), (0,1), (2,0), (0,2), (1,1)]$$

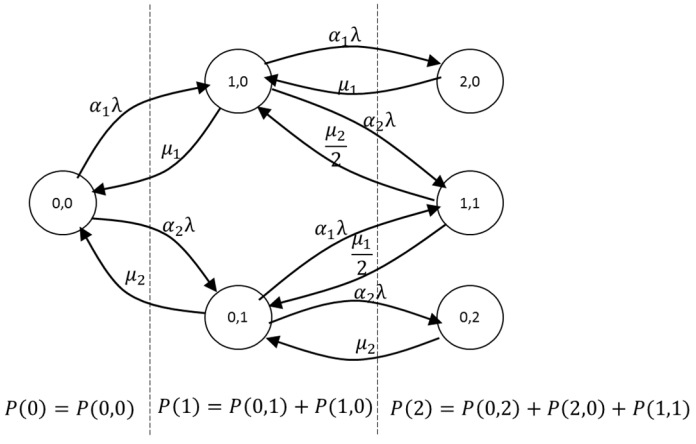


Figure 7. CPU time-sharing task workload state diagram

Since there is no dependency between the quantum CPU times, all inter-quantum times are exponentially distributed. Therefore, there is an embedded Markov Chain. The balance equations are as following:

$$\lambda p(0,0) = \mu_1 p(1,0) + \mu_2 p(0,1);$$

$$(\mu_1 + \lambda)p(1,0) = \mu_1 p(2,0) + \lambda\alpha_1 p(0,0) + \frac{\mu_2}{2} p(1,1);$$

$$(\mu_2 + \lambda)p(0,1) = \mu_2 p(0,2) + \lambda\alpha_2 p(0,0) + \frac{\mu_1}{2} p(1,1);$$

$$\mu_1 p(2,0) = \lambda\alpha_1 p(1,0); \qquad \mu_2 p(0,2) = \lambda\alpha_2 p(0,1);$$

$$\frac{\mu_1 + \mu_2}{2} p(1,1) = \lambda\alpha_1 p(0,1) + \lambda\alpha_2 p(1,0)$$

By solving the equations, we have:

$$p(1,0) = \frac{\lambda\alpha_1}{\mu_1} p(0,0); \quad p(0,1) = \frac{\lambda\alpha_2}{\mu_2} p(0,0);$$

$$p(0,2) = (\frac{\lambda\alpha_2}{\mu_2})^2 p(0,0); \qquad p(2,0) = \left(\frac{\lambda\alpha_1}{\mu_1}\right)^2 p(0,0);$$

$$p(1,1) = 2\frac{\lambda\alpha_1}{\mu_1}\frac{\lambda\alpha_2}{\mu_2} p(0,0)$$

Since,

$$P(0) = p(0,0); \; P(1) = p(0,1) + p(1,0); \qquad P(2) = p(2,0) + p(0,2) + p(1,1);$$

Then the solved model is as following:

$$P(1) = \lambda\left(\frac{\alpha_1}{\mu_1} + \frac{\alpha_2}{\mu_2}\right)p(0); \qquad P(2) = (\lambda(\frac{\alpha_1}{\mu_1} + \frac{\alpha_2}{\mu_2}))^2 p(0).$$

This result is as the same as the solution of M/M/1/FCFS queue. It seems that the result of M/M/1/FCFS can be used to solve M/M/1/PS model approximately. This hypothesis needs to be proved in a more general situation, such as $n$ map tasks sharing a CPU resource with the quantum oriented Round Robin discipline, where $0 < n < \infty$.

***Proof.*** Let $n$ map tasks sharing the CPU resource, each task perceiving CPU to be slower by a factor $n$ due to the quantum size trending to zero, then the state space is as following:

$$\{(n_1, n_2): n_1, n_2 \geq 0, n = n_1 + n_2, 0 \leq n \leq \infty\}$$

Where $n_1$ is number of the map task in the phase 1 and $n_2$ is number of the map task in the phase 2, and $n = n_1 + n_2$ for the total number of the map tasks in CPU queue. For the probability $P(n_1, n_2)$ of tasks in CPU being in state $(n_1, n_2)$, we have the equations for total rate leaving and entering the state as following:

$$Total\ rate\ leaving\ state\ (n_1, n_2) = \begin{cases} \lambda P(n_1, n_2) & task\ leaving\ CPU \\ \frac{\mu_1 n_1}{n_1 + n_2} P(n_1, n_2) & task\ leaving\ phase\ 1 \\ \frac{\mu_2 n_2}{n_1 + n_2} P(n_1, n_2) & task\ leaving\ phase\ 2 \end{cases}$$

$$Total\ rate\ entering\ state\ (n_1, n_2)$$

$$= \begin{cases} \lambda\alpha_1 P(n_1 - 1, n_2) & task\ entering\ phase\ 1 \\ \lambda\alpha_2 P(n_1, n_2 - 1) & task\ entering\ phase\ 2 \\ \frac{\mu_2(n_2 + 1)}{n_1 + n_2 + 1} P(n_1, n_2 + 1) + \frac{\mu_1(n_1 + 1)}{(n_1 + n_2 + 1)} P(n_1 + 1, n_2) & new\ task\ entering\ CPU \end{cases}$$

From observing the previous mathematical derivation of solving the balance equations, the hypothesis solution is as following:

$$P(n_1, n_2) = \binom{n_1 + n_2}{n_1} \rho_1^{n_1} \rho_2^{n_2} P(0,0) \quad where\ \rho_1 = \frac{\lambda \alpha_1}{\mu_1}, \rho_2 = \frac{\lambda \alpha_2}{\mu_2}, \alpha_1 + \alpha_2 = 1 \quad (4.0)$$

In order to justify this hypothesis, we substitute the equation (4.0) into the total rate entering states $(n_1, n_2)$, and get the equations as following:

$$task\ entering\ phase\ 1 = \lambda \alpha_1 \binom{n_1 + n_2 - 1}{n_1 - 1} \rho_1^{n_1 - 1} \rho_2^{n_2} P(0,0)$$

$$= \binom{n_1 + n_2}{n_1} \rho_1^{n_1} \rho_2^{n_2} P(0,0) \frac{\lambda \alpha_1}{\rho_1} \frac{n_1}{n_1 + n_2} = P(n_1, n_2) \frac{\mu_1 n_1}{n_1 + n_2} = task\ leaving\ phase\ 1$$

$$task\ entering\ phase\ 2 = \lambda \alpha_2 \binom{n_1 + n_2 - 1}{n_1} \rho_1^{n_1} \rho_2^{n_2 - 1} P(0,0)$$

$$= \binom{n_1 + n_2}{n_1} \rho_1^{n_1} \rho_2^{n_2} P(0,0) \frac{\lambda \alpha_2}{\rho_2} \frac{n_1}{n_1 + n_2} = \frac{\mu_2 n_1}{n_1 + n_2} P(n_1, n_2) = task\ leaving\ phase\ 1$$

$$new\ task\ entering\ CPU$$

$$= \binom{n_1 + n_2 + 1}{n_1} \rho_1^{n_1} \rho_2^{n_2 + 1} P(0,0) \frac{\mu_2 (n_2 + 1)}{n_1 + n_2 + 1}$$

$$+ \binom{n_1 + n_2 + 1}{n_1 + 1} \rho_1^{n_1 + 1} \rho_2^{n_2} P(0,0) \frac{\mu_1 (n_1 + 1)}{(n_1 + n_2 + 1)}$$

$$= \binom{n_1 + n_2}{n_1} \rho_1^{n_1} \rho_2^{n_2} P(0,0) \frac{n_1 + n_2 + 1}{n_2 + 1} \rho_2 \frac{\mu_2 (n_2 + 1)}{n_1 + n_2 + 1}$$

$$+ \binom{n_1 + n_2}{n_1} \rho_1^{n_1} \rho_2^{n_2} P(0,0) \frac{n_1 + n_2 + 1}{n_1 + 1} \rho_1 \frac{\mu_1 (n_1 + 1)}{(n_1 + n_2 + 1)}$$

$$= (\lambda \alpha_1 + \lambda \alpha_2) P(n_1, n_2) = \lambda P(n_1, n_2) = task\ leaving\ CPU$$

The equations clearly show that the hypothesis solution is one of the right solutions. From solving the balance equations in Figure 7, it also proved that this solution is the unique solution. Then let's solve the total probability $P(n)$ of state $(n_1, n_2)$.

Since $n = n_1 + n_2$, with Binomial theorem [26, page 306-309], we have:

$$\mathrm{P}(n) = \sum_{n_1=0}^{n} P(n_1, n - n_1) = \sum_{n_1=0}^{n} \binom{n}{n_1} \rho_1^{n_1} \rho_2^{n-n_1} P(0,0) = (\rho_1 + \rho_2)^n P(0,0)$$

Since $\rho_1 + \rho_2 = \frac{\lambda\alpha_1}{\mu_1} + \frac{\lambda\alpha_2}{\mu_2} = \lambda \sum_{i=1}^{2} \frac{\alpha_i}{\mu_i}$ and $P(0) = P(0,0)$, the solution is as:

$$\mathrm{P}(n) = \lambda^n \left( \sum\nolimits_{i=1}^{2} \frac{\alpha_i}{\mu_i} \right)^n P(0),$$

Let's solve the case that is even more general: $m$ quantum with $n$ map tasks sharing a CPU. Base on the pervious solution, there is a hypothesis as following:

$$P(n_1, n_2, \dots, n_m) = \binom{n}{n_1 n_2 \dots n_m} \rho_1^{n_1} \rho_2^{n_2} \dots \rho_m^{n_m} P(0,0)$$

$$where\ \rho_1 = \frac{\lambda\alpha_1}{\mu_1}, \rho_2 = \frac{\lambda\alpha_2}{\mu_2}, \rho_m = \frac{\lambda\alpha_m}{\mu_m}, \sum_{i=1}^{m} a_i = 1, n = n_1 + \cdots + n_m$$

By substituting the above equation into all the entering rates of state $(n_1, n_2, \dots, n_m)$ and CPU, the equations show that all the entering rates of state $(n_1, n_2, \dots, n_m)$ and CPU equal to all the leaving rates of state $(n_1, n_2, \dots, n_m)$ and CPU. Therefore, it proved that this equation is one of the right solutions and is the unique solution in the Markov Chain.

Since,

$$P(n) = \sum_{n_1+\cdots+n_m=n} P(n_1, n_2, \ldots, n_m) = \sum_{n_1+\cdots+n_m=n} \binom{n}{n_1 n_2 \ldots n_m} \rho_1^{n_1} \rho_2^{n_2} \ldots \rho_m^{n_m} P(0,0)$$

By Multinomial theorem [26, Page 310-318], we have:

$$P(n) = \lambda^n \left( \sum_{i=1}^{m} \frac{\alpha_i}{\mu_i} \right)^n P(0,0)$$

Let $\frac{1}{\mu} = \sum_{i=1}^{m} \frac{\alpha_i}{\mu_i}$, $\sum_{i=1}^{n} \alpha_i = 1$ and $\rho = \frac{\lambda}{\mu}$, we have: $P(n) = \rho^n (1-\rho)$ (4.1)

Observing the equation (4.1), it is the same as the solution of M/M/1/FCFS queueing model .That proves that we can use the model of M/M/1/FCFS to approximately solve the departure model of the map or reduce task, which is running on a multiuser operating system. ■

According to the workflow of Map-Reduce, the input file size will affect the departure time of a Map-Reduce job. A bigger size of input file will generate a larger number of map tasks. Thanks to the fact that the limiting probability of M/G/1/PS is independent of the job size distribution (it depend only on its mean) and it is called an insensitivity property; see [22, Page 389-390]. Therefore, the previous solution can also be used to solve the Map-Reduce job with different input file sizes

It is necessary to add up all the map/reduce tasks' completion time within a Map-Reduce job, since the job departure time is Map-Reduce job service time instead of a signal task's completion time. The time for serving the total map tasks in a job is $\sum_{i=1}^{n} \tau_i^m$ and the time for completing all reduce tasks in a job is $\sum_{i=1}^{n} \tau_i^r$.

Since the stationary time series, a constant ensemble mean of completion time for a job in map phase is as following:

$$T_{job-m} = \lim_{t\to\infty} \mu(t) = \lim_{m\to\infty} \frac{\sum x_t}{m} = \lim_{N_{job}\to\infty} \frac{\sum_{i=1}^{n} \tau_i^m}{N_{job} p_m} \quad where \quad n \le \infty;\ 0 \le \tau_i^m < \infty$$

, and the independent character of the completion time of the map task, $\sum_{i=1}^{n} \tau_i^m$ in interval $(0, t]$ can be formulated as a recursive equation $= h(t) + \int_0^t h(t-u)n(u)du$.

Therefore, we have:

$$\sum_{i=1}^{n} \tau_i^m = \lim_{t\to\infty} \int_0^t T_m(t-u)dN_m(u) = \int_0^\infty T_m(\infty)dN_m(u) = T_m(\infty)N_m(\infty)$$

$$T_{job-m}p_m = \frac{\sum_{i=1}^{n} \tau_i^m}{N_{job}} = \frac{T_m(\infty)N_m(\infty)}{N_{job}} = T_m^{avg} N_m; \qquad p_m = N_m^{slot}$$

where $T_m(t) = \int_0^t xt_m(x)dx$ is expected completion time of map tasks; $N_m(t)$ is expected number of completed map tasks (renewal function; see [25 Page 62]) in time $t$;

$T_i^m =$ The execution time of $i$th map task;

$N_m =$ The average number of map tasks for a job;

$T_m^{avg} =$ The average execution time of a map task;

$N_m^{slot} =$ The number of configured map slots.

The mean of Map service time for a job and rate of job (map phase) are as:

$$T_{job-m} = \frac{T_m^{avg} \times N_m}{N_m^{slot}}; \qquad \mu_m = \frac{1}{T_{job-m}}$$

A constant mean of completion time for a job in reduce phase is as following:

$$T_{job-r} = \lim_{t\to\infty} \mu(t) = \lim_{N_{job}\to\infty} \left(\frac{\sum_{i=1}^{n1} \tau_i^r}{N_{job} p_r} + \frac{\sum_{j=1}^{n2} \tau_j^{sh}}{N_{job} p_r}\right) where\ \ n1, n2 \le \infty;\ 0 \le \tau_i^r, \tau_j^{sh} < \infty$$

and

$$\frac{\sum_{i=1}^{n1} \tau_i^r}{N_{job}} = \frac{1}{N_{job}} \lim_{t\to\infty} \int_0^t T_r(t-u) dN_r(u) = \frac{T_r(\infty) N_r(\infty)}{N_{job}} = T_r^{avg} N_r$$

$$\frac{\sum_{j=1}^{n2} \tau_j^{sh}}{N_{job}} = \frac{1}{N_{job}} \lim_{t\to\infty} \int_0^t T_{sh}(t-u) dN_{sh}(u) = \frac{T_{sh}(\infty) N_{sh}(\infty)}{N_{job}} = T_{sh}^{avg} N_{sh}$$

where $T_r(t), N_r(t), T_{sh}(t), N_{sh}(t)$ functions of expected completion time of tasks and expected number of completed tasks in time $t$;

$T_i^r,\ T_j^{sh}$ = The execution time of $i$th reduce sub-task, $j$th shuffle sub-task;

$N_r,\ N_{sh}$ = The average number of reduce sub-tasks for a job, shuffle sub-tasks for a job;

$T_r^{avg},\ T_{sh}^{avg}$ = The average execution time of a reduce sub-task, a shuffle sub-task;

$N_r^{slot}$ = The number of configured reduce slots.

We have Mean of Reduce service time and rate of job (reduce phase) as:

$$T_{job-r} = \frac{T_r^{avg} \times N_r}{N_r^{slot}} + \frac{T_{sh}^{avg} \times N_{sh}}{N_r^{slot}}; \qquad \mu_r = \frac{1}{T_{job-r}};$$

By Map-Reduce job workload, every map task has a map phase and every reduce task has shuffle and reduce phases. Since the size of the data block can be configured as a fixed

size of chunks for the map phase with the same data searching algorithm and the standard hardware performance, the average execution time for the map task ($T_m^{avg}$) is determined. Since number of shuffle sub-tasks is depend on the results of the map tasks and number of reduce sub-tasks is depend on the merger results of the shuffle sub-tasks, with the same data sorting algorithm and the standard disk data access and CPU performance, the average execution time for shuffle sub-task ($T_{sh}^{avg}$) and reduce sub-task ($T_r^{avg}$) are determined too. The configurable parameter of number slots in parallel ($N^{slot}$) is used to determine that thread number $p_m$ and $p_r$ for parallel running the map tasks and reduce tasks respectively. Therefore, the rate of the completion time $\mu_m$ and $\mu_r$ are determined.

Since the reduce task depends on the result of the map task, it is necessary to put these two different kinds of tasks into a renewal process with the sequence order. The departure time of a Map-Reduce job cannot be exponentially distributed. However, by separating the job departure time into two individual phases and connecting them in a series, the state transition diagram (Figure 8) shows us that there is an embedded Markov Chain.

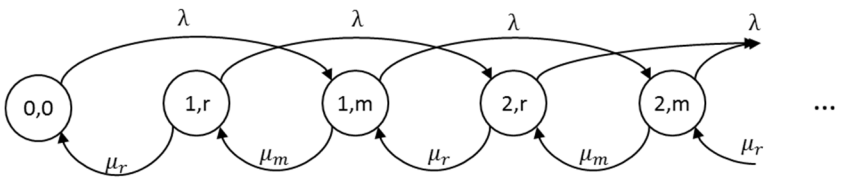


Figure 8 Map-Reduce job departure states diagram

Figure 8 also clearly shows us the departure model of Map-Reduce job is a typical hypo-exponential distribution. Then we conclude the departure model as:

$$F_x(X) = \begin{cases} 1 - \dfrac{\mu_r e^{-\mu_m x}}{\mu_r - \mu_m} - \dfrac{\mu_m e^{-\mu_r x}}{\mu_m - \mu_r} & where \;\; \mu_m \neq \mu_r \\ 1 - e^{-\mu_m x}(1 + \mu_m x) \;\; where \;\; \mu_m = \mu_r \;\; (Erlang\ 2) \end{cases}$$

$$\overline{E(T)} = \frac{1}{\mu} = \frac{1}{\mu_m} + \frac{1}{\mu_r}; \qquad where \;\; \rho = \lambda\left(\frac{1}{\mu_m} + \frac{1}{\mu_r}\right) < 1$$

Since the departure model is a hypo-exponential distribution that combines several exponential distributions in a sequence order, there is an embedded Markov Chain. Therefore, the whole service time of Map-Reduce job is a PH distribution that can be viewed as the distribution of the time until absorption in suitably defined Markov processes; see [25, Page 61-79], and it is fully qualified to map this job departure model onto the Markov Chain.

### 3.3 Repair Process

The node repair process is defined on our availability model where it can be considered as a birth-death process [23, Page 365-387]. It is used to predict the degree to which a worker node cluster in a specified operable and committable state at the start of a mission when the mission is called for at a random time interval $(0, t]$. It provides the ratio of total time the worker node cluster is capable of being used during a given interval to the length of the interval.

Map-Reduce provides scalable, fault-tolerant and rack-aware worker node designed to be deployed on commodity hardware [17]. It is designed with hardware failure in mind and built for large datasets, with a fixed length of data block. It is optimized for sequential operations and rack-aware cross-platform as well as heterogeneous cluster [17]. Data in a Map-Reduce is broken down into smaller units (called blocks) and distributed throughout

the cluster. Each block is duplicated twice (for three copies), with the two replicas stored on two worker nodes in a rack somewhere else in the cluster [27]. Since the data has a default replication factor of three, it is a relatively higher available and fault-tolerant. If a copy is lost (because of a worker node failure, for example), Map-Reduce will automatically replicate it elsewhere in the cluster, ensuring that the three fold replication factor is maintained [27]. In Map-Reduce, the data is transmitted in the form of small packages between the worker nodes, which improves the throughput of data access [17]. Figure 9 shows the worker node and the data failover process. In the node repair procvess, the nodes can be repaired when they are broken. After the node is repaired completely, the node will be restored to be as good as new.

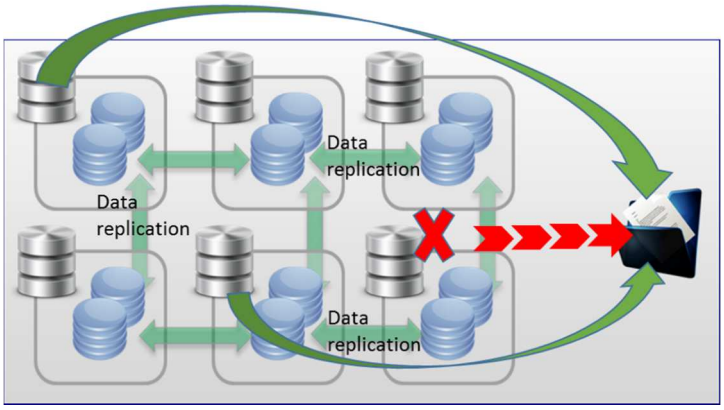


Figure 9 Worker node data replication and failover processes

### 3.4 The Availability Model

Since we are interested in that the worker nodes are running on a stable situation, we assume that the failure rate $g(t)$ of the worker node is a constant value that is independent of the worker node age $t$. Then we have the failure rate of the worker node $f$. We also assume that any two of the worker nodes would not fail at the exactly

same time. Moreover, any failure of the worker node would not affect the failure of others. With the previous assumptions and the same idea that we have proved in the job arrival model, it can be sure that the arrival of failed worker nodes is a Poisson distribution.

Since the worker nodes are running in the cluster and all the "spare" nodes are in active model, the failover time is relatively much small. Therefore, we have the reason to assume that the switching time that the tasks from failure node move to the "active" spare node is zero in this paper.

In case of the low bound of node repairing performance, we assume that the repair rate will keep in the lowest performance. Therefore, the repair rate is a constant value $r$. Since the repair time of the worker node would not depend on the repairing history of other failed worker nodes due to the lowest repair performance for every broken node, the repair time is exponential distribution for sure.

Since the node repair time is exponential distribution and inter-node failure time is exponential distributed too, the transition of the nodes failure and repairing states can be mapped to the Markov Chain. Since all good nodes will fail and all failure nodes can be repaired and back to work by our assumption, this Markov Chain should be irreducible. Since the total number of the worker nodes should be countable, we have the total number of the worker nodes as a finite number $S$. We have the worker nodes failure and repairing state diagram as following:

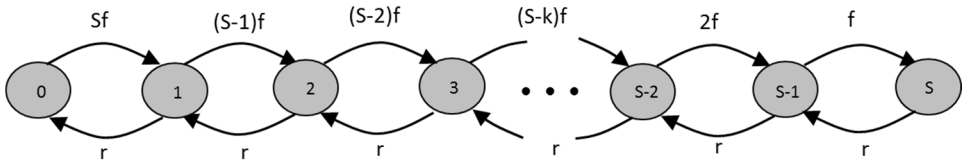


Figure 10 Worker node failure and repairing state diagram

Since this state diagram is an irreducible Markov Chain, total rate leaving state $i$ = total rate entering state $i$. Then we have the balance equations as following:

$$SfP(0) = rP(1)$$

$$P(k) = \frac{(S-k)f}{r}P(k-1)$$

$$rP(S) = fP(S-1) \quad where \quad 0 < k < S$$

The equations solved, we have:

$$P(k) = \prod_{k=0}^{S-1} \frac{(S-k)f}{r} P(0)$$

$$= \left(\frac{f}{r}\right)^k \frac{S!}{(S-k)!} P(0) \quad where \quad 0 \le k \le S$$

For normalizing the equation:

$$P(0) + P(1) + \cdots + P(k) + \cdots + P(S) = 1$$

Then we have:

$$P(0) \sum_{k=0}^{S} \frac{S!}{(S-k)!} \left(\frac{f}{r}\right)^k = 1,$$

$$P(0) = \frac{1}{\sum_{k=0}^{S} \frac{S!}{(S-k)!} (\frac{f}{r})^k}$$

$$P(k) = \left(\frac{f}{r}\right)^k \frac{S!}{(S-k)!} \left( \frac{1}{\sum_{k=0}^{S} \frac{S!}{(S-k)!} (\frac{f}{r})^k} \right)$$

$$= \frac{\left(\frac{f}{r}\right)^k \frac{S!}{(S-k)!}}{\sum_{k=0}^{S} \frac{S!}{(S-k)!} (\frac{f}{r})^k} \qquad (5.1)$$

Availability of the worker nodes in Map-Reduce cluster is as following:

$$A = \sum_{k=0}^{S-1} P_{failure}(k) = 1 - P_{failure}(s) = 1 - P_{node}(0) = 1 - \frac{\left(\frac{f}{r}\right)^s s!}{\sum_{k=0}^{S} \frac{s!}{(s-k)!} \left(\frac{f}{r}\right)^k}$$

$$where \quad P(k) = P_{failure}(k) = P_{node}(S-k)$$

The proposed availability model of Map-Reduce computing is a 3-tuple state space $(i, j, k)$, where $i$ is the number of Map-Reduce job; $j$ indicates the phases of the job (map phase, reduce phase); and $k$ is the number of failure nodes such that $0 \leq k \leq S$, where $S$ the maximum possible number of the worker nodes in a Map-Reduce computing. There are spare nodes that can replace any failed node associated with either a job in a map phase or in a reduce phase. It is assumed that Map-Reduce computing is a multi-server (node) queueing model, and all the Map-Reduce jobs are served in FCFS schedule. Each state is represented by a random variable $q_{i,j,k}$.

A 3-dimension state transition diagram is designed as Figure 11, with a state definition of $\{i, j, k\}$ with a state space of $(0,1,2, \dots, m) \times (0,1) \times (0,1,2, \dots S)$ where $0 < m, 0 < S$ and $m, S$ are countable. For every element in this state transition diagram, $P(i, j, k) = P(I = i, J = j, K = k)$ denotes a random event that there are $i$ number of Map-Reduce jobs in the queue; $j$ presents the phase type (0: map; 1: reduce); and $k$ is the number of failure nodes while one of the job in map/reduce phase is being served at a long instant of time $t = t_1$.

Since we already proved the solutions and assumed the parameters in the job arrival model and the job departure model as well as node repair process, then all parameters used in our availability model can be defined as follows:

1. $\lambda$: job arrival rate; it follows a Poisson distribution.
2. $\mu_m, \mu_r$: each virtual task service rate for the job map phase and reduce phase, respectively and their inverse, that is, the times, are exponentially distributed.
3. $f$: each node failure rate; it follows a Poisson distribution.
4. r: each node repair rate for the node repair and the repair time is exponential distribution

Then, we conclude that the following states transition of Map-Reduce computing has all the properties that a typical Markov Chain has:

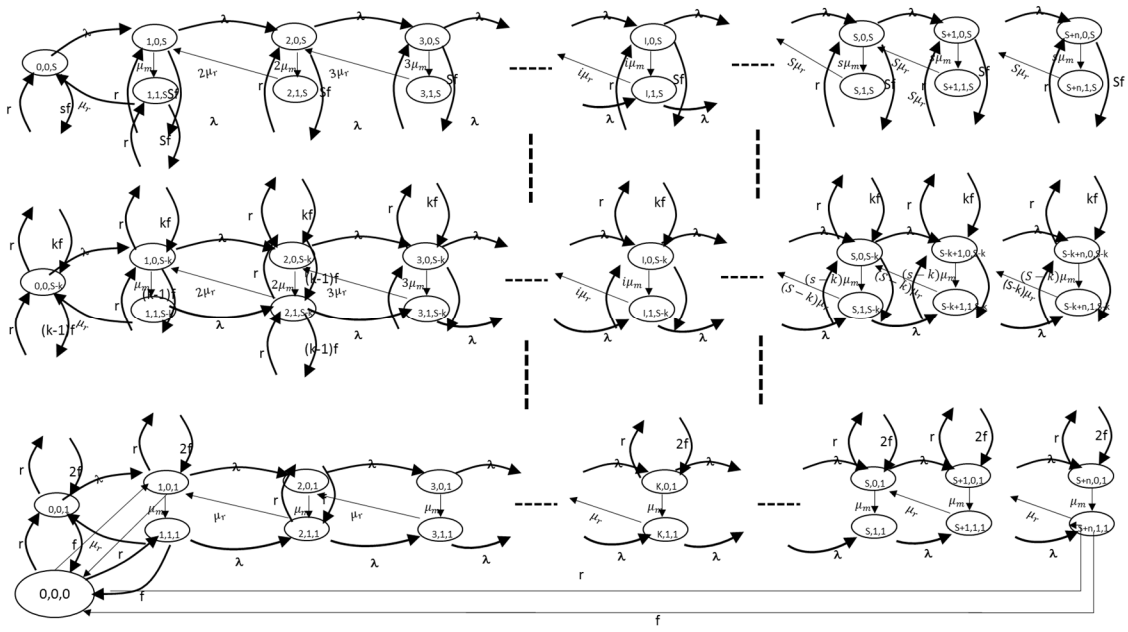


Figure 11 State diagram of availability Map-Reduce computing

### 3.5 Solving the Model

From the state diagram, we observe that there are several embedded Markov chains in there. Therefore, we have the balance equations as following:

$$(\lambda + Sf)P(0,0,S) = \mu_r P(1,1,S) + rP(0,0,S-1) \quad (6.1)$$

$$(\mu_m + \lambda + Sf)P(1,0,S) = \lambda P(0,0,S) + 2\mu_r P(2,1,S) + rP(1,0,S-1) \quad (6.2)$$

$$(\mu_r + \lambda + Sf)P(1,1,S) = \mu_m P(1,0,S) + rP(1,1,S-1) \quad (6.3)$$

$$(2\mu_m + \lambda + Sf)P(2,0,S) = \lambda P(1,0,S) + 3\mu_r P(3,1,S) + rP(2,0,S-1) \quad (6.4)$$

$$(i\mu_m + \lambda + Sf)P(i,0,S) = \lambda P(i-1,0,S) + (i+1)\mu_r P(i+1,1,S) + rP(i,0,S-1)$$

$$where\ 3 \le i \le S-1 \quad (6.5)$$

$$(S\mu_m + \lambda + Sf)P(i,0,S) = \lambda P(i-1,0,S) + S\mu_r P(i+1,1,S) + rP(i,0,S-1)$$

$$where\ S-1 \le i \qquad (6.6)$$

$$(\lambda + r)P(0,0,1) = \mu_r P(1,1,1) + 2fP(0,0,2) \qquad (6.7)$$

$$(\mu_m + \lambda + r)P(1,0,1) = \lambda P(0,0,1) + \mu_r P(2,1,1) + 2fP(1,0,2) \qquad (6.8)$$

$$(\mu_r + \lambda + r)P(1,1,1) = \mu_m P(1,0,1) + 2fP(1,1,2) \qquad (6.9)$$

$$(\mu_m + \lambda + r)P(2,0,1) = \lambda P(1,0,1) + \mu_r P(3,1,1) + 2fP(2,0,2) \qquad (6.10)$$

$$(\mu_m + \lambda + r)P(i,0,1) = \lambda P(i-1,0,1) + \mu_r P(i+1,1,1) + 2fP(i,0,2)$$

$$where\ 3 \le i \le S-1 \qquad (6.11)$$

$$(i\mu_m + \lambda + (S-k)f + r)P(i,0,S-k)$$
$$= \lambda P(i-1,0,S-k) + (i+1)\mu_r P(i+1,1,S-k) + rP(i,0,S-k-1)$$
$$+ (S-k+1)f(i,0,S-k+1)$$

$$(i\mu_r + \lambda + (S-k)f + r)P(i,1,S-k)$$
$$= \lambda P(i-1,1,S-k) + i\mu_m P(i,0,S-k) + rP(i,0,S-k-1)$$
$$+ (S-k+1)f(i,0,S-k+1)$$

$$where \quad 3 \le i \le S-k \ \ and \ \ 0 < k \le S-2 \qquad (6.12)$$

$$((S-k)\mu_m + \lambda + (S-k)f + r)P(i,0,S-k)$$
$$= \lambda P(i-1,0,S-k) + (S-k)\mu_r P(i+1,1,S-k) + rP(i,0,S-k-1)$$
$$+ (S-k+1)f(i,0,S-k+1)$$

$$((S-k)\mu_r + \lambda + (S-k)f + r)P(i,1,S-k)$$

$$= \lambda P(i-1,1,S-k) + (S-k)\mu_m P(i,0,S-k) + rP(i,0,S-k-1)$$

$$+ (S-k+1)f(i,0,S-k+1)$$

$$where\ \ S-k \le i\ \ and\ \ 0 < k \le S-2 \qquad (6.13)$$

Since the probability failure of the worker nodes is independent to the probability of the number of Map-Reduce jobs in a Map-Reduce computing, $q_{i,j,k}(t)$ can be modeled as a joint probability. Hence, a series product of two independent probabilities can be assumed: one is the probability of Map-Reduce jobs that are served in either map phase or reduce phase on the condition of a certain number of nodes (e.g., $S-k$); another is the probability to have the same number of nodes(e.g., $S-k$) in the system such that

$$P(i,j,k) = P(i,j) * P_{server}(S-k) = P(i,j) * P_{failure}(k).$$

In order to solve $P(i,j)$, we have the following balance equations:

When the number of nodes is $S$:

$$\lambda P(0,0) = \mu_r P(1,1) \qquad (6.a1)$$

$$(\mu_m + \lambda)P(1,0) = \lambda P(0,0) + 2\mu_r P(2,1) \qquad (6.a2)$$

$$(\mu_r + \lambda)P(1,1) = \mu_m P(1,0) \qquad (6.a3)$$

$$(2\mu_m + \lambda)P(2,0) = \lambda P(1,0) + 3\mu_r P(3,1) \qquad (6.a4)$$

$$(i\mu_m + \lambda)P(i,0) = \lambda P(i-1,0) + (i+1)\mu_r P(i+1,1)$$

$$(i\mu_r + \lambda)P(i,1) = \lambda P(i-1,1) + i\mu_m P(i,0)$$

$$where\ 3 \leq i \leq S-1 \qquad (6.a5)$$

$$(S\mu_m + \lambda)P(i,0) = \lambda P(i-1,0) + S\mu_r P(i+1,1)$$

$$(S\mu_r + \lambda)P(i,1) = \lambda P(i-1,1) + S\mu_m P(i,0)$$

$$where\ \ S-1 \leq i \qquad (6.a6)$$

When the number of nodes is 1:

$$\lambda P(0,0) = \mu_r P(1,1) \qquad (6.a7)$$

$$(\mu_m + \lambda)P(1,0) = \lambda P(0,0) + \mu_r P(2,1) \qquad (6.a8)$$

$$(\mu_r + \lambda)P(1,1) = \mu_m P(1,0) \qquad (6.a9)$$

$$(\mu_m + \lambda)P(2,0) = \lambda P(1,0) + \mu_r P(3,1) \qquad (6.a10)$$

$$(\mu_m + \lambda)P(i,0) = \lambda P(i-1,0) + \mu_r P(i+1,1)$$

$$(\mu_r + \lambda)P(i,1) = \lambda P(i-1,1) + \mu_m P(i,0)$$

$$where\ \ 0 \leq i \qquad (6.a11)$$

When $k$ number of nodes have failed:

$$(i\mu_m + \lambda)P(i,0) = \lambda P(i-1,0) + (i+1)\mu_r P(i+1,1)$$

$$(i\mu_r + \lambda)P(i,1) = i\mu_m P(i,0) + \lambda P(i-1,1)$$

$$where\ 3 \leq i \leq S-k-1\ and\ \ 0 < k \leq S-2 \qquad (6.a12)$$

$$((S-k)\mu_m + \lambda)P(i,0) = \lambda P(i-1,0) + (S-k)\mu_r P(i+1,1)$$

$$((S-k)\mu_r+\lambda)P(i,1)=\lambda P(i-1,1)+(S-k)\mu_m P(i,0)$$

$$where\ \ S-k\le i and\ 0<k\le S-2 \qquad (6.a13)$$

From (6.a1) to (6.a3), we have:

$$P(1,1)=\frac{\lambda}{\mu_r}P(0,0)$$

$$P(1,0)=\frac{(\mu_r+\lambda)}{\mu_m}P(1,1)=\frac{\lambda(\mu_r+\lambda)}{\mu_r\mu_m}P(0,0)$$

$$P(2,1)=\frac{(\mu_m+\lambda)P(1,0)}{2\mu_r}-\frac{\lambda P(0,0)}{2\mu_r}$$

From (6.a7) to (6.a11), we have:

$$P(1,1)=\frac{\lambda}{\mu_r}P(0,0)$$

$$P(1,0)=\frac{(\mu_r+\lambda)}{\mu_m}P(1,1)=\frac{\lambda(\mu_r+\lambda)}{\mu_r\mu_m}P(0,0)$$

$$P(2,1)=\frac{(\mu_m+\lambda)P(1,0)}{\mu_r}-\frac{\lambda P(0,0)}{\mu_r}$$

From (6.a12) to (6.a13), we have:

$$P(i,0)=\begin{cases}\dfrac{(i\mu_r+\lambda)}{i\mu_m}P(i,1)-\dfrac{\lambda}{i\mu_m}P(i-1,1) & where\ \ 3\le i\le S-k-1;\\ & 0<k\le S-2\\ \dfrac{((S-k)\mu_r+\lambda)}{(S-k)\mu_m}P(i,1)-\dfrac{\lambda}{(S-k)\mu_m}P(i-1,1) & where\ \ S-k\le i;\\ & 0<k\le S-2\end{cases}$$

$$P(i,1) = \begin{cases} \dfrac{(i\mu_m + \lambda)}{(i+1)\mu_r} P(i-1,0) - \dfrac{\lambda}{(i+1)\mu_r} P(i-2,0) & where \;\; 3 \le i \le S-k-1; \\ & 0 < k \le S-2 \\ \dfrac{\big((S-k)\mu_m + \lambda\big)}{(S-k)\mu_r} P(i-1,0) - \dfrac{\lambda}{(S-k)\mu_r} P(i-2,0) & where \;\; S-k \le i; \\ & 0 < k \le S-2 \end{cases}$$

Note that it is without loss of generality, assumed that there is no dependency between Map-Reduce jobs and the worker node failures as proved by substituting the solutions $P(i,0)$ and $P(i,1)$ into equations from (6.1) to (6.13), where it can be shown that those equations also hold.

The boundary probability $P(m,0), P(m,1)$ as following:

$$P(m,0) = \begin{cases} \dfrac{\lambda}{m\mu_m} P(m-1,0) & where \;\; m \le S-k-1; \; 0 < k \le S-2 \\ \dfrac{\lambda}{(S-k)\mu_m} P(m-1,0) & where \;\; S-k \le m; \; 0 < k \le S-2 \end{cases}$$

$$P(m,1) = \begin{cases} \dfrac{(m-1)\mu_m + \lambda}{m\mu_r} P(m-1,0) - \dfrac{\lambda}{m\mu_r} P(m-2,0) & where \;\; m \le S-k-1; \\ & 0 < k \le S-2 \\ \dfrac{\big((S-k)\mu_m + \lambda\big)}{(S-k)\mu_r} P(m-1,0) - \dfrac{\lambda}{(S-k)\mu_r} P(m-2,0) & where \;\; S-k \le i; \\ & 0 < k \le S-2 \end{cases}$$

Since $P(i,0)$ and $P(i,1)$ are the probabilities of $i$ number of Map-Reduce jobs with $(S-k)$ number of available nodes in the system, where $k$ the total number of failed nodes (can be denoted in this thesis by$P_{server}(S-k)$ $or$ $P_{failure}(k)$), by normalization, we obtain the following:

$$P(0,0) + \sum_{i=1}^{m} \big(P(i,0) + P(i,1)\big) = P_{server}(S-k) = P_{failure}(k)$$

$$P(0,0) = P_{failure}(k) - \sum_{i=1}^{m} \big(P(i,0) + P(i,1)\big)$$

Note that as $P(i,0)$ and $P(i,1)$ are the probabilities under the condition of $k$ number of failed nodes,

$$\frac{P(0,0) + \sum_{i=1}^{m} \big(P(i,0) + P(i,1)\big)}{P_{failure}(k)} = 1$$

Then, the following holds:

$$P(0,0) = P_{failure}(k) - \sum_{i=1}^{m} \big(P(i,0) + P(i,1)\big)$$

Since we already solved $P(k) = P_{failure}(k)$ in node resilience model (5.1) as follows:

$$P(k) = P_{failure}(k) = P_{node}(S-k) = \frac{\left(\frac{f}{r}\right)^k \times \frac{S!}{(S-k)!}}{\sum_{k=0}^{S} \frac{S!}{(S-k)!} (\frac{f}{r})^k}$$

We have:

$$P(0,0) = P_{node}(S-k) - \sum_{i=1}^{m} \big(P(i,0) + P(i,1)\big)$$

Finally, we solve the 3-tuple analytical availability model as following:

$$P(i,j,k) = \begin{cases} P(i,0)P_{node}(S-k) \\ P(i,1)P_{node}(S-k) \end{cases} \quad where \;\; 0 \le i < m, \quad 0 \le k \le S, \quad j \in (0,1)$$

$$P(m,j,k) = \begin{cases} P(m,0)P_{node}(S-k) \\ P(m,1)P_{node}(S-k) \end{cases} \quad where \;\; 0 < m, \quad 0 \le k \le S, \quad j \in (0,1)$$

$$where \qquad \rho = \frac{\lambda(\mu_m + \mu_r)}{(S-k)\mu_m\mu_r} < 1 \qquad (6.14)$$

Using equation (6.14) and little's law, we can obtain the useful functions as following:

1. Mean of throughput (departure rate) in the cluster where number of the worker nodes $= S - k$:

$$\bar{E} = \sum_{i=1}^{m} \mu_r P(i,1,k) = \sum_{i=1}^{m} \mu_r P(i,1) \frac{(\frac{f}{r})^k \frac{s!}{(s-k)!}}{\sum_{k=0}^{s} \frac{s!}{(s-k)!}(\frac{f}{r})^k}$$

$$where \ \ 0 \le i < m, \qquad 0 \le k \le S \ and \ \frac{\lambda(\mu_m + \mu_r)}{(S-k)\mu_m\mu_r} < 1$$

2. Mean of jobs in the cluster where number of the worker nodes $= S - k$:

$$\bar{N} = \sum_{i=1}^{m} (iP(i,0,k) + (m-i)P(m-i,1,k))$$

$$= \sum_{i=1}^{m} (iP(i,0) + (m-i)P(m-i,1)) \frac{(\frac{f}{r})^k \frac{s!}{(s-k)!}}{\sum_{k=0}^{s} \frac{s!}{(s-k)!}(\frac{f}{r})^k}$$

$$where \ \ 0 \le i < m, \qquad 0 \le k \le S \ \ and \ \frac{\lambda(\mu_m + \mu_r)}{(S-k)\mu_m\mu_r} < 1$$

3. Mean of the turnaround time:

$$\bar{T} = \frac{\bar{N}}{\bar{E}} = \frac{\sum_{i=1}^{m} \big(iP(i,0) + (m-i)P(m-i,1)\big)}{\sum_{i=1}^{m} \mu_r P(i,1)}$$

$$where \ \ 0 \le i < m, \qquad 0 \le k \le S \ and \ \frac{\lambda(\mu_m + \mu_r)}{(S-k)\mu_m\mu_r} < 1$$

# CHAPTER IV

## SIMULATIONS

Parametric simulations have been conducted in order to observe the trends of the availability and the impact of the availability on the performance, particularly throughput and turnaround time, based on the proposed analytical model.

It is assumed that the maximum number of spare nodes ($S$) is set to 10 for practical purposes (note that simulation results become invisible beyond $S$= 10 as far as any change in availability trend is concerned, in other words due to convergence), and $m$=500, $\lambda = 3.0/\text{s}, \mu_m = 3.5/\text{s}, \mu_r = 3.5/\text{s}$. Five different failure/repair (i.e., $\frac{f}{r}$) ratios are simulated such as 0.1, 0.5, 0.8, 1.0 and 2.

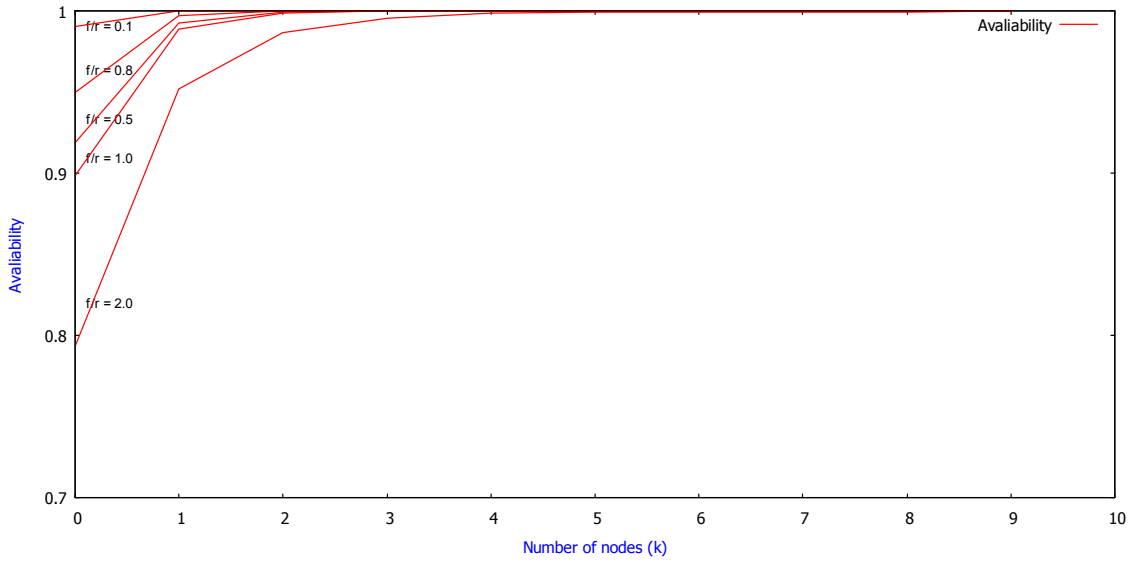


Figure 12 Trends of availability

The plots above demonstrate the availability trends versus the number of available nodes (i.e., $k$) at different $\frac{f}{r}$ ratios. Just as expected the availability with higher $\frac{f}{r}$ ratios picks up more quickly than the ones with lower $\frac{f}{r}$ ratios at a given $k$ value. Further, it is observed that as the value of $k$ increases the gap between availability plots narrows. This implies this simulation repair process has more significant effect as fewer number of nodes are engaged and available. However, notice that there is no bouncing point observed other than convergence. It might be possibly argued that there is a missing variable or the availability is somehow dragged down when $k$ value picks up.

In order to look at the impact of the cost of availability (i.e., the higher availability exercised the higher cost of resources required) on the performance, a throughput analysis has been conducted and the results are shown as follows.

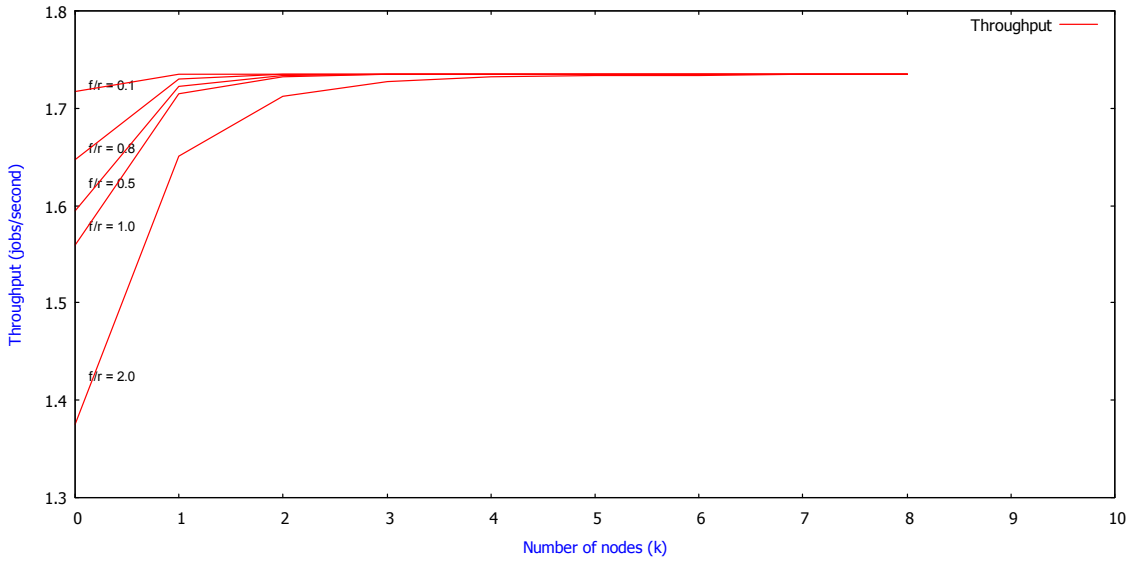


Figure 13 Trends of throughput

With $k$ as a common variable in the two independent probability terms in the proposed availability model, it is demonstrated that the throughput trends concur with the variable $k$ when it increases. This is made possible due to the repair process to pick up failed nodes back into the loop.

In order to further observe the direct impact of availability on the throughput at different $\frac{f}{r}$ ratios, the following plots are drawn with $k$ = 0, 1, 3 and 5. Each plot in the graph below demonstrates the throughput trends versus their corresponding availability at various $k$ values

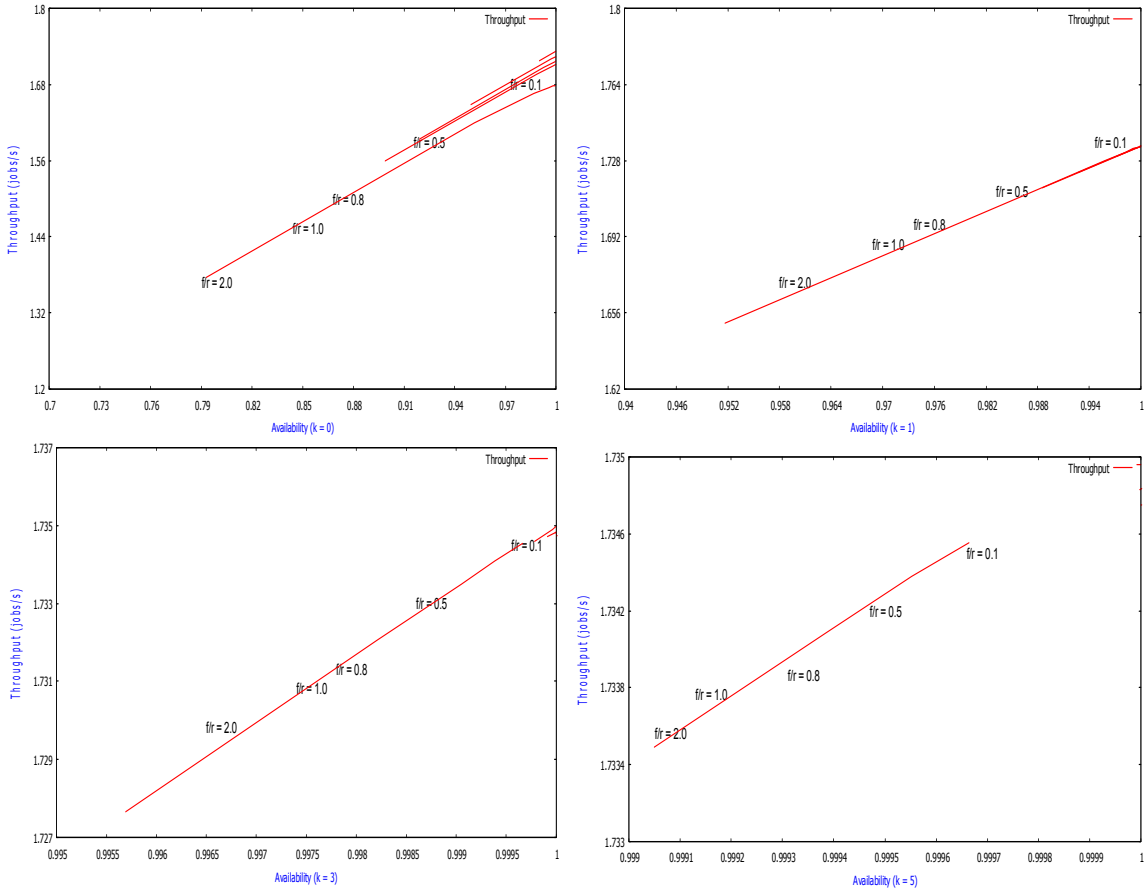


Figure 14 Throughput versus availability trends

The plot above reveals the trends of throughput versus the availability trends with different number of failure nodes (i.e., $k$). It is observed that when $k = 0$, the availability stretches out from about 0.79 to converge 1.0 and the throughput stretches out from 1.38 to 1.72 as $\frac{f}{r}$ stretches from 2.0 down to 0.1 (specifically, the slope, i.e., (Δthroughput/Δavailability), is 1.62); when k = 1, the availability from 0.952 to 1.0, and the throughput from 1.65 to 1.73 ((Δthroughput/Δavailability), is 1.666); when $k = 3$, the availability from 0.9954 to 1.0, and the throughput from 1.7275 to 1.7345 ((Δthroughput/Δavailability), is 1.52); and when k = 5, the availability from 0.9995 to

1.0, and the throughput from 1.7335 to 1.7345 ((Δthroughput/Δavailability), is 2.0). Notice that the trends are inconsistent as far as the slopes are concerned and as the $k$ value picks up from 0, 1, 3 to 5, the trends are neither monotonic increase nor decrease.

In order to further observe the impact of availability on the job turnaround time at different map/reduce service time $\mu_m, \mu_r$ ratios, the following plots are drawn with number of nodes in the cluster from 0 to 10. Each plot in the graph below demonstrates the turnaround time trends versus their corresponding number of the worker nodes values.

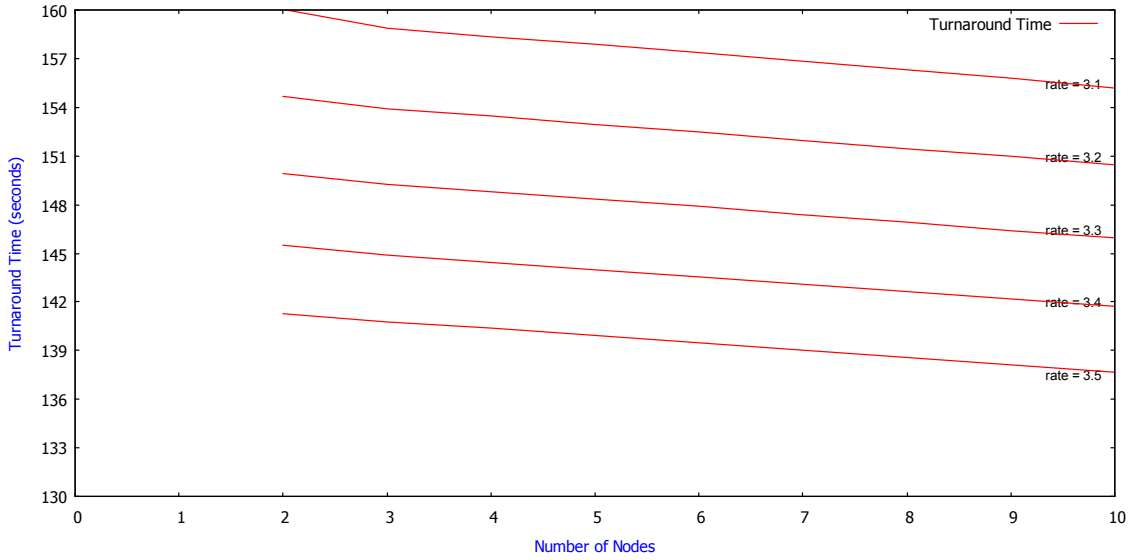


Figure 15 Trends of the job turnaround time

The plots above demonstrate the turnaround time trends versus the number of the worker nodes at different $\mu_m, \mu_r$ ratios. It is observed that the turnaround time will linearly decrease by increasing number of the worker nodes in Map-Reduce computing. However, the slop (Δturnaround/Δk) is (141-137.8)/8 = 0.4 for the service rates $\mu_m, \mu_r = 3.5/s$. In other words, when the number of worker nodes begin to saturate, for example, two nodes

are already be able to keep the system in a stable ($\frac{\lambda(\mu_m+\mu_r)}{(S-k)\mu_m\mu_r} = \frac{2\lambda\mu_r}{(10-8)\mu_r{}^2} = \frac{\lambda}{\mu_r} = \frac{3.0}{3.1} < 1$) in our cases, adding more spare worker nodes could not give us much more performance improvement. In this case where each of the rates is 3.5/s, we only gain the performance improvement ratio by 0.3 (0.4/average turnaround time=137.8x4x0.4 ≈ 0.3%) per node. From Figure 12, we also know that the availability has almost reached 100% when the number of nodes $k$ is 3 and the repairing speed is faster than the node failure. Therefore, it seems not worth providing the number of spare nodes more than three in order to improve the performance 0.3%/node in this case. Maybe this conclusion can explain that Hadoop Map-Reduce framework suggested the default replication factor to be 3 rather than 4 or 10.

# CHAPTER V

## CONCLUSION AND DISCUSSIONS

This paper has presented an analytical model to evaluate the availability and performance of Map-Reduce computing on a Hadoop architecture.

The proposed model involves a set of variables, simulation of the number of Map-Reduce jobs, and the number of the worker nodes engaged as well as a few constants such as job arrival, and departure rates, node failure and repair rates. The proposed availability model provides a comprehensive yet theoretical basis to assure and optimize the design of Map-Reduce computing in particular terms of availability and with reference to the performance (particularly, throughput, turnaround time) as well.

Parametric simulations have been conducted and it demonstrates efficacy in assessing the availability, and it has been observed that the availability with higher $\frac{f}{r}$ ratios picks up more quickly than the ones with lower $\frac{f}{r}$ ratios at a given $k$ value; the throughput trends concur with the availability trends as the $k$ value (the number of available nodes) picks up. This is made possible by the repair process to pick up failed nodes back into the loop; and the trends are inconsistent as far as the slopes ($\Delta$throughput/$\Delta$availability) are concerned and as the $k$ value picks up from 0, 1, 3 to 5. The last parametric simulation is

performed for obtaining the job turnaround time at different map/reduce service time $\mu_m, \mu_r$ ratios and various number of the worker nodes. The results show that it may not be worthwhile to prepare more than two spare nodes for backing up a worker node in the cluster.

## REFERENCES


[1] K. Salah and R. Boutaba, ”Estimating Service response Time for Elastic Cloud Applications”, *IEEE1st International Conference on Cloud Networking*, 2012.

[2] ”Bigdata High Availability (HA) Architecture”, Blazegraph (white paper).

[3] Jiang, Li; Xu, Qiang; Eklow, Bill, ”On effective TSV repair for 3D-stacked ICs”, *Design, Automation& Test in Europe Conference & Exhibition (DATE)*, 2012

[4] Ljung, Lennart. ”Model error modeling and control design.” (2000).

[5] Huang, Tian, David J. Whitehouse, and Derek G. Chetwynd. ”A unified error model for tolerancedesign, assembly and error compensation of 3-DOF parallel kinematic machines with parallelogramstruts.” *CIRP Annals-Manufacturing Technology 51.1* (2002): 297-301.

[6] Shin, Kang G. ”Error Detection Process? Model, Design, and Its Impact on Computer Performance.” *IEEE transactions on computers 6* (1984): 529-540.

[7] Wasi-ur-Rahman, Md, Xiaoyi Lu, NusratSharmin Islam, Raghunath Rajachandrasekar, and DhabaleswarK. Panda. ”High-Performance Design of YARN MapReduce on Modern HPC Clusters withLustre and RDMA.” In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*,pp. 291-300. IEEE, 2015.

[8] Yu, Xiaolong, and Wei Li. ”Performance modeling and analysis of mapreduce/hadoop workloads.” *In Local and Metropolitan Area Networks (LANMAN)*, *2015 IEEE International Workshop on*, pp. 1-6.IEEE, 2015.

[9] Chao, Shen, Tong Weiqin, and SaminaKausar. ”Predicting the Performance of Parallel ComputingModels using Queuing System.” In *Cluster, Cloud and Grid Computing (CCGrid)*, *2015 15th IEEE/ACM International Symposium on*, pp. 757-760. IEEE, 2015.

[10] Lin, Jian, Fan Liang, Xiaoyi Lu, Li Zha, and Zhiwei Xu. ”Modeling and Designing Fault-ToleranceMechanisms for MPI-Based MapReduce Data Computing Framework.” In *Big Data Computing Serviceand Applications (BigDataService)*, *2015 IEEE First International Conference on*, pp. 176-183.IEEE, 2015.

[11] Lin, Chi-Yi, Ting-Hau Chen, and Yi-No Cheng. ”On Improving Fault Tolerance for HeterogeneousHadoop MapReduce Clusters.” In *Cloud Computing and Big Data (CloudCom-Asia)*, *2013 InternationalConference on*, pp. 38-43. IEEE, 2013.

[12] Wang, Hao, Haopeng Chen, and Fei Hu. ”ReCT: Improving MapReduce performance under failureswith resilient checkpointing tactics.” In *Big Data (Big Data)*, *2014 IEEE International Conference on*, pp. 27-32. IEEE, 2014.

[13] Marynowski, Joo Eugenio, Altair OlivoSantin, and Andrey Ricardo Pimentel. ”Method for testing thefault tolerance of MapReduce frameworks.” *Computer Networks* (2015).

[14] Dogra, Naveen, and Sarbjeet Singh. ”A Survey of Dynamic Replication Strategies in Distributed Systems.” *International Journal of Computer Applications 110.11* (2015).

[15] Ahuja, Mini Singh, Randeep Kaur, and Dinesh Kumar. ”Trend Towards the Use of Complex Networksin Cloud Computing Environment.” *International Journal of Hybrid Information Technology8.3* (2015).

[16] R. Lammel, “Google’s Map-Reduce programming model—Revisited,” *Sci. Comput. Program., vol. 70, no. 1*, pp. 1-30, 2008

[17] Apache Hadoop[Online]. Available: http://hadoop.apache.org./

[18] J. Dean and S. Ghemawat. “Map-Reduce Simplified data processing on large clusters,” *Commun. ACM, vol. 51. No. 1*, pp. 107-113, 2008

[19] G. Bolch S. Greiner and H, Meer, *Queueing networks and Markov chain*, 1st edn, Wiley, 1998

[20] E. Vianna G. Comarela and T. Pontes, “Analytical Performance Model for Map-Reduce Workloads” *Int J parallel Prog (2013)* 41:495-525.

[21] C. Paul and M. Andrew, *Introduction Time Series with R*, Springer, 2009.

[22] M. Harchol-Balter, *Performance modeling and design of computer systems*, Cambridge University Press, 2013

[23] K. S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall, inc., 1982

[24] CISCO Troubleshooting Ethernet Collisions[Online] Available: https://www.cisco.com/c/en/us/support/docs/interfaces-modules/port-adapters/12768-eth-collisions.html

[25] G. Latouche and V. Ramaswami, *Introduction to matrix Analytic Methods in Stochastic Modeling*, ASA-SIAM. Philadelphia, 1999

[26] J. Gallier, *Discrete Mathematics*, Springer, 2011

[27] Hadoop achitechural overview [Online]. Avaliable: https://www.datadoghq.com /blog/hadoop-architecture-overview/

[28] V. Paxson and S. Floyd, "Wide-area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, pp.226-244, June 1995.

VITA

Zuqiang Ke

Candidate for the Degree of

Master of Science

Thesis: AVAILIBILITY MODELING AND ASSURANCE OF MAP-REDUCE COMPUTING

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Bachelor of Engineering in computer science and technology at Shanghai Jiao Tong University, Shanghai, China in September, 1987.

Completed the requirements for the Master of Science in computer science at Oklahoma State University, Stillwater, Oklahoma in December, 2017.