AVAILABILITY OF JOBTRACKER MACHINE IN HADOOP/MAPREDUCE ZOOKEEPER COORDINATED CLUSTERS

Ekpe Okorafor[1] and Mensah Kwabena Patrick[2]

[1]Department of Computer Science, African University of Science and Technology, Abuja, Nigeria
eokorafor@aust.edu.ng
[2]Department of Computer Science, University for Development Studies, Navrongo, Ghana
menkpat@yahoo.com

ABSTRACT

It is difficult to use the traditional Message Passing Interface (MPI) approach to implement synchronization, coordination, and prevent deadlocks in distributed systems. This difficulty is lessened by the use of Apache's Hadoop/MapReduce and Zookeeper to provide Fault Tolerance in a Homogeneously Distributed Hardware/Software environment. A mathematical model for the availability of the JobTracker in Hadoop/MapReduce using Zookeeper's Leader Election Service is presented in this paper. Although the availability is less than what is expected in f+1 Fault Tolerance systems for crash failures, this approach makes coordination and synchronization easy, reduces the effect of Byzantine faults and provides Fault Tolerance for distributed systems. The results obtained show that the availability changes with change in the number of Zookeeper servers. This model can help determine how many servers are optimal for high availability, from which vendor they must be purchased, and when to use a Zookeeper coordinated Hadoop cluster to perform safety critical tasks.

KEYWORDS

Hadoop, Byzantine fault, JobTracker, Markov model, Availability

1. INTRODUCTION

A major challenge facing organizations today is the ability to organize and process large data generated by customers. According to Nielson Online [1] there are more than 1 billion internet users. How much data these users are generating and how it is processed largely determines the success of the organization concerned. It was a similar demand to process large datasets in Google that inspired Engineers to introduce MapReduce [2]. MapReduce is designed to run on commodity nodes (cheaper machines) that can fail at any time. Its performance does not reduce significantly due to network latency. It exhibits high fault tolerance and is easy to use by programmers who have no prior experience in parallel programming. Apache's Hadoop [3] is an open source implementation of Google's MapReduce. It is made up of MapReduce and Hadoop Distributed File System (HDFS). Hadoop/MapReduce is currently implemented as Master-Slave architecture; this makes both the Hadoop Distributed File System Master node (NameNode) and the MapReduce Master Node (JobTracker) single point of failures. The failure of a Slave node (DataNode or TaskTracker) does not pose serious challenge since the Master node simple re-assigns tasks that were to be processed by the failed node to another node. Researchers have proposed several solutions to the availability issue of the JobTracker in Hadoop/MapReduce. Among these solutions is the use of Apache's Zookeeper [4] as a

DOI : 10.5121/acij.2012.3302

coordinating service when implementing Hadoop/MapReduce. This implementation provides automatic failover mechanism for the JobTracker by redirecting JobClients and TaskTrackers to the new active JobTracker in case the current active JobTracker goes down. In this paper, a Markov model is used to determine how available the JobTracker is in a Zookeeper coordinated Hadoop/MapReduce cluster.

2. DESCRIPTION OF HADOOP/MAPREDUCE AND ZOOKEEPER

2.1. Hadoop

Hadoop [5] is an open source framework which allows parallel programmers to write and run distributed applications that process large amounts of data. It is an open source implementation of Google's MapReduce. Hadoop is implemented with the capability to run MapReduce programs written in Java, C++, Python, Ruby, etc. on top of the Hadoop Distributed File System (HDFS). MapReduce is then used to divide a user application into small blocks which are then replicated and stored on multiple nodes in the cluster by means of the HDFS. These applications are executed in parallel by MapReduce on individual nodes that are assigned a Map or Reduce task. Hadoop is made up of several daemons. These daemons are the major constituents of the Hadoop framework as shown in Figure 1.

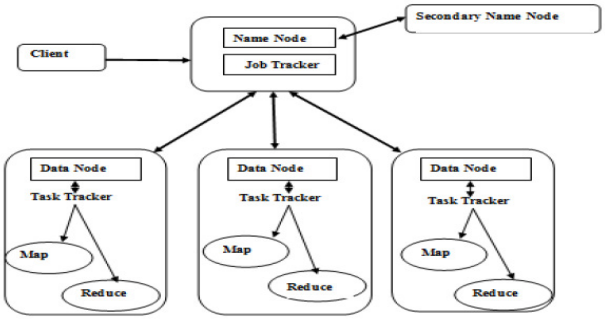

Figure 1.The Hadoop Framework

NameNode: The Name Node is the Master of the Distributed File System. Its task is to direct the slave DataNode daemons on how to carry out low level input/output (I/O) tasks. The name node monitors and controls the storage, use, and health of the HDFS in a cluster. It keeps track of the file metadata; i.e. which files are currently in the system and how each file is broken down into file blocks. The task of the NameNode is I/O and memory intensive; hence the machine on which it resides does not double as a DataNode and a NameNode at the same time.
Secondary NameNode: Each Hadoop cluster contains one Secondary Name Node that serves as an assistant daemon to the NameNode. It constantly monitors the NameNode and keeps a checkpoint of HDFS metadata at given intervals. When the NameNode fails, the file system can be recovered from the checkpoints of the Secondary NameNode.
DataNode: The Data Node daemon is hosted by each slave machine in the cluster. It reads and writes the HDFS blocks to actual files on the local file system. If a client wants to read or write to the HDFS, it must ask for the location (i.e. which DataNode is hosting that file block) from the NameNode, and then communicates directly with that DataNode. It is also possible for Data Nodes to communicate with each other when replication of data is needed for redundancy. Upon initialization, each DataNode reports the status of its file system blocks to the NameNode, after that, the DataNode constantly polls the NameNode to inform it of new changes and to also receive instructions on how to proceed with delete, modify, or write instructions that will affect local persistent storage.

JobTracker: The JobTracker daemon works in Master-Slave architecture. It is a single point of failure in a typical Hadoop framework. Client jobs are submitted to the JobTracker (as jar files). It is the responsibility of the JobTracker to determine which job must be executed first (default uses FIFO), which task to assign to which machine, and must monitor the progress of the execution on each of the slave machines. Each slave node must periodically send a heartbeat signal to the JobTracker machine. If a Master does not receive a heartbeat from a slave, it assumes the slave is down and must consequently re-launch an instance of a TaskTracker on a different machine and submit the same job that failed. This is transparent to the user and must also auto-scale.

2.2. MapReduce

MapReduce is used to divide user applications into small blocks which are then replicated and stored on multiple nodes in the cluster by means of HDFS. Google implements MapReduce as shown in Figure 2. The principles behind the implementation are well documented in [2].

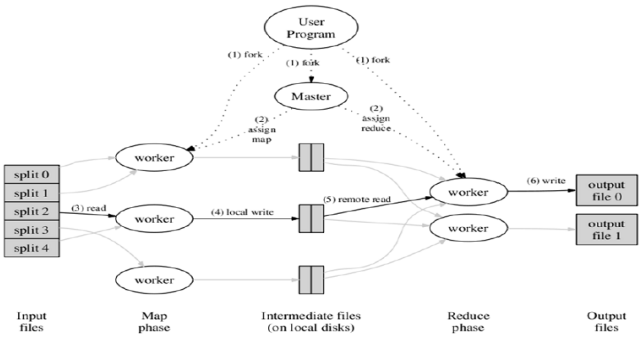

Figure 2. Google's implementation of MapReduce [2]

2.3. Zookeeper and Zookeeper Leader Election Framework

ZooKeeper is a high-performance, available and scalable open-source software package [6] that is used to coordinate distributed systems. It is a stripped-down file system that exposes some primitives on which distributed systems can build to implement higher level services such as synchronization, naming, and configuration management. It is difficult to implement coordination in distributed systems especially if these coordinating services are to be written from scratch. ZooKeeper is organized in a hierarchical tree of nodes called znodes. Znodes are created by clients. Each znode can have children and data associated with it. The size of the data is limited to 1MB per node [7] since ZooKeeper nodes are not meant for data storage but to keep data in memory in order to achieve high throughput and low latency. However, there are no renames, soft or hard links and no append semantics. All znodes can have children except ephemeral znodes that disappears as soon as the client that created it closes its session. A client may manipulate a regular node by creating and/or deleting it explicitly, as opposed to ephemeral nodes where the client can either decide to delete the node or let the service remove it automatically when the client's session expires or a failure occurs (this property is the basis for implementing Zookeeper Leader Election protocol; see [4]). All nodes are seen by all clients, and apart from ephemeral nodes, all nodes can be deleted by any other client including the client that created the node. Zookeeper guarantees the following [7]:

Atomicity: A client must either have access to the entire data stored at a node or it will obtain nothing at all. This design is to maintain consistency.

Sequential Consistency: Zookeeper uses the First in First out (FIFO) policy to execute request that update the state of the service.
Single System Image: No matter which server a client connects to, zookeeper guarantees that they see the same data.
Reliability and Availability: Zookeeper is designed to work on multiple machines with the ability of a client to reconnect to another server should the original server serving the client go down.
Timeliness: Updates and other operations in zookeeper are propagated to all concerned servers and clients in real time.
Programmers take advantage of the capabilities of Zookeeper's API to implement some important algorithms such as Leader Election used to manage distributed systems. It is important that when a Leader goes down, a Follower rises to the position of a new Leader to carry on the processing of client requests. For each zookeeper cluster, there must be 2f+1 server to enable it to achieve failover; since zookeeper is fault-tolerant only if majority of servers are up and running (f is the number of servers whose failure the cluster can tolerate). One of the servers is made active during start-up and is said to be the Leader. All other servers are Followers. Nodes register to the Leader Election Service to enable them get notification when a server goes down, and as soon as this happens, a quorum is formed, and a new Leader is elected. It is important to get a Leader because any write request from clients can only be processed by the Leader, however, read requests can be processed by any of the zookeeper servers. The elected Leader waits for Followers to connect to it, it will then sync with them by sending any updates they are missing. Zookeeper is highly fault tolerant. The failure of a client does not impact negatively on its performance. Such a client will always try to reconnect until it is successful. There are two possibilities involved when a Zookeeper server fails: First, if the failed server is the Leader in a Leader Election Service, then the rest of the servers must come together to elect a new leader. This is only possible if the working Zookeeper servers form a majority. In the second case, if the failed server is a Follower, the Leader will try to connect with it, if this fails; the Leader assumes that particular Follower is down, the Leader Election Service must then determine either the rest of the servers form a majority. If not, the entire Zookeeper service will go down. This implies that, the only time a Zookeeper coordinated cluster will go down aside unforeseen circumstances is when the rest of the working servers do not form a majority of the servers in the cluster.

2.4. Related Work

In his 2008 presentation, Francesco Salbaroli [8] proposed a Fault-Tolerant Hadoop JobTracker in which he advocates the addition of a library (JGroups) to the Hadoop source code to aid in making the JobTracker highly available. His proposal aimed at maintaining the current Master-Slave implementation of Hadoop/MapReduce since it has a relatively small overhead and reduced coordination complexities. Though this implementation adds a little overhead (negligible due to the number of replicated JobTrackers), it can accommodate new features without modifying its current behaviour. The logical model of the replicated JobTracker is made up of a coordinating protocol that resides on the Distributed File System (DFS), and performs coordination between replicated JobTrackers and Slave machines. JGroups perform the discovery of members, health checking, implementation of election protocol, and communication between components. If a master fails, a Fault Tolerant Manager discovers its failure and triggers a new election accordingly. Devarajulu K. [4] proposed an implementation of Hadoop/MapReduce that will make the JobTracker highly available. His implementation was based on the Leader Election Framework suggested in Zookeeper. In the Zookeeper Leader Election framework, 2f + 1 zookeeper servers are started with only one of them acting as the Master and the rest Followers. This service can tolerate the failure of f servers since the remaining servers must form a majority. Oliviera et al [9] proposed the development of a

Markov based model for the analysis of the availability of a telecommunications management system. The model was used to define the availability of parts of the system, identifying hardware and software components responsible for reducing the availability of the system, and to define actions that can mitigate the low availability. Laprie [10] proposed a dependability model for software systems during their operational life. He used a Markov model for a single machine that considered the dependability of both software and hardware on the system. His model considered the system as being partitioned into a software part and a hardware part, each of which can fail and can be repaired. Dai et al [11] proposed a model for determining the reliability and availability of centralized heterogeneous systems. They presented a model that implements a system availability function for a virtual machine along with an application example to illustrate their method and to demonstrate its feasibility. Lai et al [12] also proposed a Markov model for determining the availability of a k Fault Tolerance (f+1) homogeneously distributed software/hardware system (HDSHS). They defined HDSHS as a distributed system in which all hosts are of the same type; such as machines from the same vendor. Their model of N homogeneous host was a cluster of one necessary host, and N-1 redundant hosts, meaning that if all of the N hosts fail the system fails; otherwise the system is functional even if only one host is active.
Cisco and Greenplum [13] in an effort to improve the usability and availability of the JobTracker have introduced a user interface and a remote mirroring that will keep a synchronized copy of cluster data at a remote site. Their MapReduce architecture provides JobTracker High Availability and Distributed NameNode High Availability to prevent job loss, "frustrating restarts, and painful failover incidents". Snapshots are kept to protect data from application errors.

In this paper, the approach of [11] is adopted, however, the model is not a cluster of one necessary host, but one which can tolerate the failure of f servers out of 2f+1 servers. This will enable the model to cater for byzantine [14] faults. In addition, Hadoop/MapReduce and Zookeeper are distributed softwares that have the properties of HDSHS if they are implemented on hardware from the same vendor. Results obtained can then be compared with those obtained in [11] to help determine the effect of Zookeeper on the availability of the cluster.

3. JOBTRACKER AVAILABILITY MODEL

The availability of a system component is the probability that the component is still operating at time t, given that it was operating at time zero. In this section, a mathematical model that is based on Markov State transitions is proposed and used to analyse the availability of the JobTracker in a Hadoop/MapReduce cluster.

3.1. Model Assumptions

1. Each machine on the cluster runs a copy of the same software. This software has a failure rate (λ_s **(t)**) determined by the Jelinski-Moranda model [15].
2. All failures in the cluster (either hardware or software) are mutually independent. If more than one failure occurs at the same time, they can either be considered as a single failure or independent failures with a time interval of zero.
3. All the machines in the cluster have hardware failure rate (λ_h) resulting from an exponential distribution.
4. A fault is corrected instantaneously without introducing new faults into the software/hardware. The correction time follows an exponential distribution with parameter μ_s for software failure and μ_h for hardware failure.
5. Both the software and hardware have only two states; working state and faulty state.

The Markov model shown in Figure 3, describes a Homogenous Distributed Hardware/Software System with N hosts [11]. Let i, j be a state where i hosts suffer hardware failure and j hosts suffer software failure. The corresponding Kolmogrov differential equation is given for i, j≠0, N; i + j ≤ N.

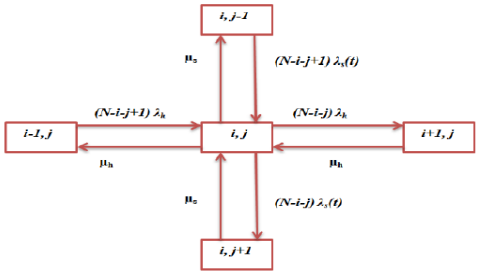

Figure 3. N-hosts HDSHS general model [11]

$$P'_{i,j}(t) = \mu_h P_{i+1,j}(t) + (N-i-j+1)\lambda_h P_{i-1,j}(t) + (N-i-j+1)\lambda_s(t) P_{i,j-1}(t) + \mu_s P_{i,j+1}(t) - X_{i,j} P_{i,j}(t) \quad (1)$$

where

$$X_{i,j} = \mu_s + (N-i-j)\lambda_h + (N-i-j)\lambda_s(t) + \mu_h$$

The initial conditions are $P_{0,0}(0) = 1$; and $P_{i,j}(0) = 0$ for i, j ≠ 0. Boundary conditions are at the following points:

$$P'_{0,0}(t) = \mu_h P_{1,0}(t) + \mu_s P_{0,1}(t) - N[\lambda_s(t) + \lambda_h] P_{0,0}(t) \quad (2)$$

$$P'_{0,j}(t) = \mu_s P_{0,j+1}(t) + \mu_h P_{1,j}(t) + (N-j+1)\lambda_s(t) P_{0,j-1}(t) - [\mu_s + (N-j)(\lambda_h + \lambda_s(t)] P_{0,j}(t) \quad (3)$$

for i=1,2,3,…,N-1.

$$P'_{i,0}(t) = \mu_s P_{i,1}(t) + \mu_h P_{i+1,0}(t) + (N-i+1)\lambda_h(t) P_{i-1,0}(t) - [\mu_h + (N-i)(\lambda_h + \lambda_s(t)] P_{i,0}(t) \quad (4)$$

for i=1,2,3,…,N-1.

$$P'_{N,0}(t) = \lambda_h P_{N-1,0}(t) - \mu_h P_{N,0}(t) \quad (5)$$

$$P'_{0,N}(t) = \lambda_s(t) P_{0,N-1}(t) - \mu_s P_{0,N}(t) \quad (6)$$

The parameter $\lambda_s(t)$ is the software failure rate. It is determined by any appropriate software reliability model and is extensively covered in [11].

3.2. Markov Model for a Hadoop/MapReduce Cluster with 3 Zookeeper Servers

Applying equations (1) to (6) on the model in Figure 4, the following Kolmogrov differential equations (7) to (16) can be obtained:

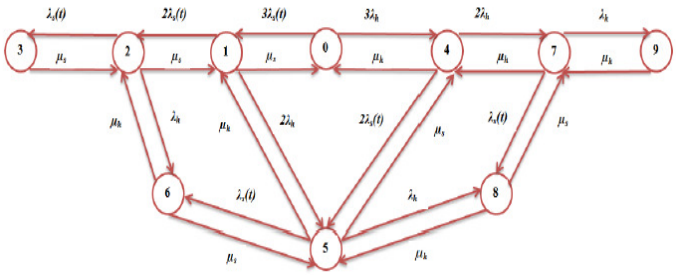

Figure 4. Markov model for N = 3 Servers

$$P_0'(t) = \mu_h P_4(t) + \mu_s P_1(t) - 3[\lambda_s(t) + \lambda_h]P_0(t) \quad (7)$$

$$P_1'(t) = \mu_s P_2(t) + \mu_h P_5(t) + 3\lambda_s(t)P_0(t) - [\mu_s + 2(\lambda_h + \lambda_s(t)]P_1(t) \quad (8)$$

$$P_2'(t) = \mu_s P_3(t) + \mu_h P_6(t) + 2\lambda_s(t)P_1(t) - [\mu_s + \lambda_h + \lambda_s(t)]P_2(t) \quad (9)$$

$$P_3'(t) = \lambda_s(t)P_2(t) - \mu_s P_3(t) \quad (10)$$

$$P_4'(t) = \mu_h P_7(t) + 3\lambda_h P_0(t) + \mu_s P_5(t) - [\mu_h + 2(\lambda_s(t) + \lambda_h]P_4(t) \quad (11)$$

$$P_5'(t) = \mu_h P_8(t) + 2\lambda_h P_1(t) + 2\lambda_s(t)P_4(t) + \mu_s P_6(t) - [\mu_s + \lambda_h + \lambda_s(t) + \mu_h]P_5(t) \quad (12)$$

$$P_6'(t) = \lambda_h P_2(t) + \lambda_s(t)P_5(t) - [\mu_s + \mu_h]P_6(t) \quad (13)$$

$$P_7'(t) = \mu_h P_9(t) + 2\lambda_h P_4(t) + \mu_s P_8(t) - [\mu_h + \lambda_s(t) + \lambda_h]P_7(t) \quad (14)$$

$$P_8'(t) = \lambda_h P_5(t) + \lambda_s(t)P_7(t) - [\mu_s + \mu_h]P_8(t) \quad (15)$$

$$P_9'(t) = \lambda_h P_7(t) - [\mu_h]P_9(t) \quad (16)$$

The system of these ten Kolmogrov differential equations (from equation (7) to (16)) can be solved numerically with the initial conditions $P_0(0) = 1$, $P_i(0) = 0$, $i = 1, 2, 3 \ldots 5$. This initial condition means that, at state 0, all hardware and software are in good working condition. Since states 0, 1, and 4 are the only working states for this model, we can determine the availability (A(t)) by solving the system of Kolmogrov differential equations using the given initial conditions.

$$A(t) = P_0(t) + P_1(t) + P_4(t) \quad (17)$$

3.3. Discussion of Results

Kolmogrov differential equations were generated for N=2, N=3, N=4 and N=5. These equations were solved numerically and the results depicted in the following plots.

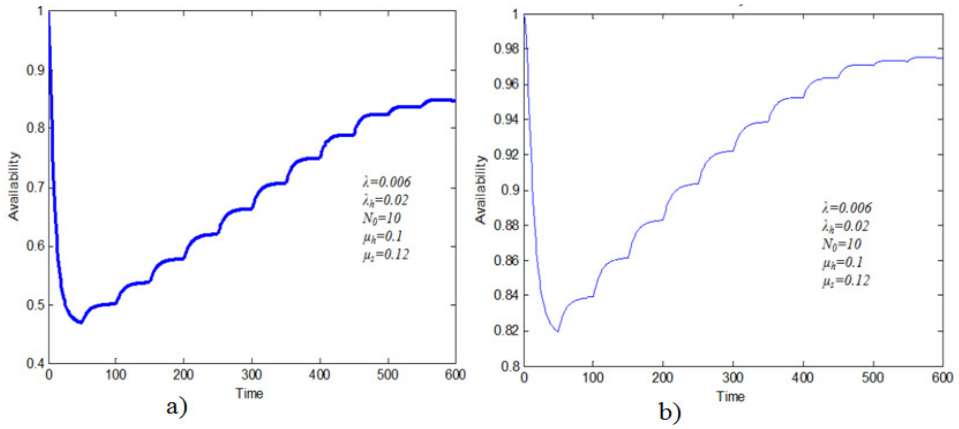

Figure 5. Availability of Zookeeper coordinated Cluster versus that of f+1 HDSHS Cluster

In Figure 5, the availability of both clusters is at its highest point immediately the cluster is started. With little time after start-up, the availability falls to the lowest point. This behaviour can be attributed to the fact that when the cluster is launched for the first time, an appreciable number of faults are detected due to say, initial start-up problems, and problems of coordination between Hadoop/MapReduce and Zookeeper, among others. At some point in time (t = 50) the availability of the cluster starts rising. This is because detected faults are fixed gradually until a point (say t > 600) when the cluster becomes bug free, causing the cluster availability to approach a certain value less than one. From these observations, it is strongly advised that the cluster must not be used to solve critical tasks before the time that it reaches its lowest availability mark. This period should be earmarked as a testing period for the cluster, possibly to help identify the number of remaining faults in the cluster.

The Zookeeper Leader Election Service requirement that only majority servers may form a quorum (2f + 1) has a negative impact on the availability of the cluster. For instance, for N=3, if two Zookeeper servers go down, the system becomes unavailable since the one remaining server does not form a majority out of the 3 servers. However, in (f+1) HDSHS systems [11], the remaining one server may continue functioning without any problem. For (f+1) HDSHS systems, the equivalent for equation (17) for N = 3 is:

$$A(t) = P_0(t) + P_1(t) + P_2(t) + P_4(t) + P_6(t) + P_7(t)$$

with a resulting availability graph shown in Figure 5b. From Figure 5, we notice that the availability for f+1 HDSHS system is higher than that of a Zookeeper coordinated Hadoop/MapReduce cluster (Figure 5a) for crash failures. However, for byzantine faults [14], the 2f+1 property of Zookeeper makes a Hadoop/MapReduce cluster highly available.

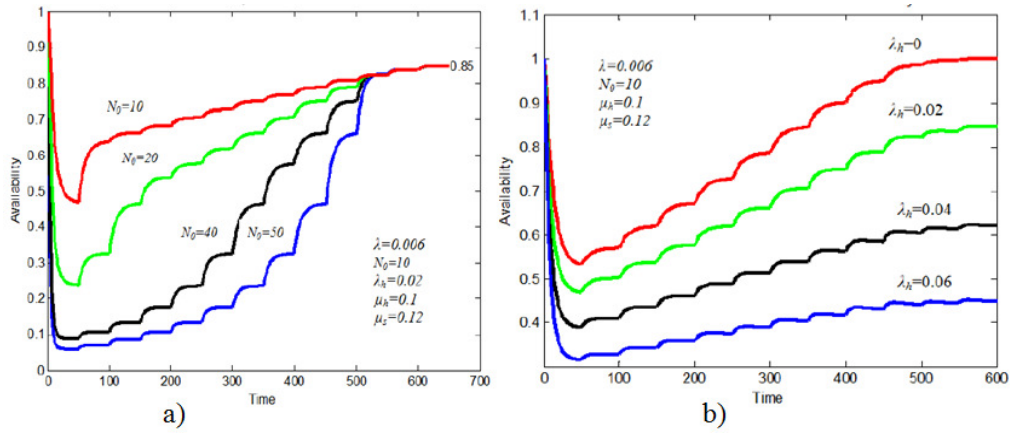

Figure 6. Effect of Initial Faults and Hardware Failure rate on Availability

For initial faults (N_0) greater than 20 (refer to Figure 6a), the availability falls below 10% at the early stages. It rises gradually, until it reaches the maximum availability mark for the cluster. It is only natural that, a system with many faults is not expected to be available until those faults are corrected. For lower initial faults, the availability rises quickly as it approaches maximum. A possible solution to this problem might be that, Zookeeper clusters should be started with a few number of servers, and the number increased gradually as we become more conversant with the type of faults that can occur due to cluster installation and management. On the other hand, each server in the cluster is assumed to originate from the same vendor and contribute λ_h as the hardware failure rate for the cluster. As the hardware failure rate increases, the resulting availability falls (Figure 6b). We can apply this in the selection of machines for the cluster, i.e., if the failure rate of machines of a particular vendor is high, they are not good choice as servers for this implementation, since for a $\lambda_h = 0$, the availability approaches 100%. This is understandable since software uses hardware as a platform to run; i.e. the failure of the hardware may result in failure of the software running on the failed hardware.

Assuming we have maintenance personnel on stand-by to correct any hardware or software faults in the cluster, then the effects of the intensity of hardware and software fault correction on

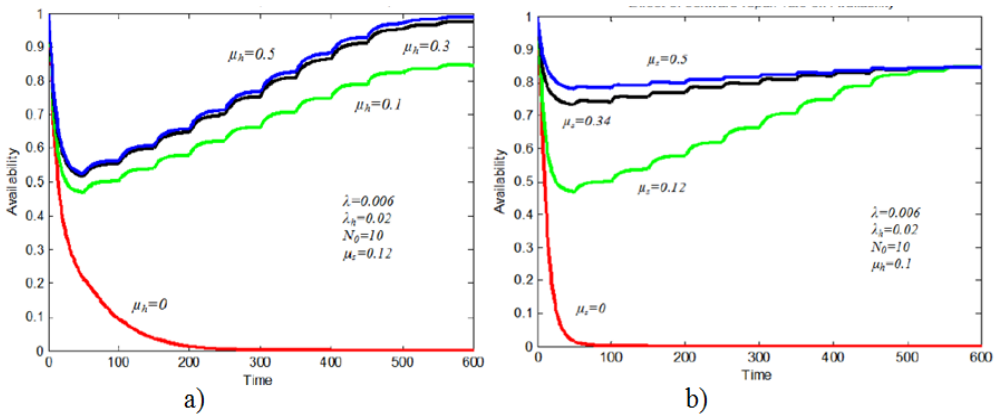

Figure 7. Effect of Software and Hardware repair rates on Availability

availability are depicted in Figures 7a and Figures 7b respectively . If the hardware faults in the cluster are left uncorrected, the availability decreases rapidly until it reaches zero; this is true for both software and hardware faults. However, as we increase the rate of maintenance of both hardware and software faults, the availability increases in response. For hardware faults, it will reach a time that the change in the rate of repair will not have much significant effect on availability since most of the hardware faults might have been removed due to regular and consistent maintenance. It should also be noted that, the effect of increasing the hardware repair rate is very significant to the behaviour of the availability curve. This can be attributed to the fact that, a highly available hardware is the first step for obtaining better functioning available software. Therefore, much of the maintenance effort must be on hardware components in the cluster. Preventive maintenance can be very useful in this sense in order to avoid unnecessary downtimes.

Figure 8 depicts the change in availability as the number of Zookeeper servers in the cluster is varied. For different values of N, the availability initially falls until it approaches t = 50 where it starts rising. For N = 2, the cluster achieves the least availability value since none of the servers is allowed to fail due to the 2f+1 property of Zookeeper. Naturally, one would expect that as the number of Zookeeper servers increase, the availability of the cluster will increase. However, this is not the case, since N = 3 has the highest availability compared to N = 4, and N = 5. This may be attributed to the fact that, with the Zookeeper service requirement of 2f + 1; out of 5 servers, 2 are allowed to fail, the remaining 3 operate as if they are a single f+1 HDSHS server, since none of them is allowed to fail in order to keep the cluster going.

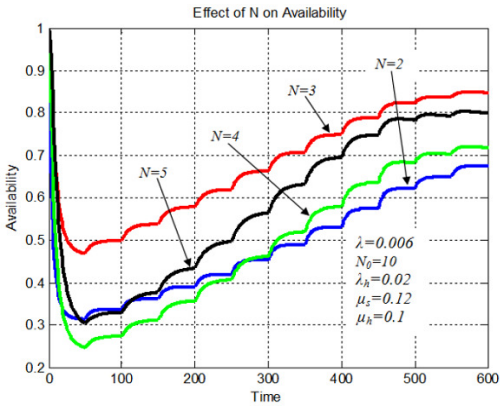

Figure 8. Effect of the Number of Servers on Availability

The 3 remaining servers have no redundancy and operate as one, hence their contribution to Availability is equivalent to that of one f+1 HDSHS server, but their contribution to both hardware and software failure rates is thrice that of a single server. This phenomenon decreases the availability of the cluster. It is therefore advisable from this model that only 3 Zookeeper servers be used when we require high availability for a Hadoop/MapReduce cluster. There is also the added advantage of reducing cost.

4. CONCLUSIONS AND FUTURE WORK

In this paper, a model to determine the availability of a Zookeeper coordinated Hadoop/MapReduce cluster was presented. Through this model, it was observed that, the availability of a Zookeeper coordinated Hadoop/MapReduce cluster is lower than that of f+1 HDSHS cluster for crash failures but higher for byzantine faults due to its 2f+1 property.

Hardware faults were observed to be critical to cluster availability for which reason they must be prioritised and fixed at the early stages of cluster life. Hardware failure rate's contribution to availability is significant, hence the suggestion that it be used as a criteria for purchasing new hardware. Finally, the model established that for a Zookeeper coordinated Hadoop/MapReduce cluster, the optimum number of servers should be 3.

REFERENCES

[1] How much data is generated on internet every year? Retrieved 14th January 2012 from http://hadoop-karma.blogspot.com/2010/03/how-much-data-is-generated-on-internet.html, and http://www.nielsen.com/us/en.html

[2] Jeffrey Dean and Sanjay Ghemawat (2004), MapReduce: Simplified Data Processing on Large Clusters, Google, Inc.

[3] Hadoop (2011). Retrieved 3rd February 2012 from http://hadoop.apache.org/

[4] Devaraju K, High Availability for JobTracker (2011). Retrieved 10th December 2011 from https://issues.apache.org/jira/browse/MAPREDUCE.

[5] Tom White (2011). Hadoop: The Definitive Guide, Second Edition, O'Reilly Media, Inc.

[6] Grant Mackey, Saba Sehrish, John Bent, Julio Lopez, Salman Habib, Jun Wang. Introducing Map-Reduce to High End Computing, University of Central Florida, Los Alamos National Lab, Carnegie Melon University.

[7] Hunt Patrick, Mahadev Konar, Flavio P. Junqueira, Benjamin Reed (2010). ZooKeeper: Wait-free coordination for Internet-scale systems. Yahoo! Grid.

[8] Francesco Salbaroli (2008), Enhancing the Hadoop MapReduce framework by adding fault tolerant capabilities; Proposal for a fault tolerant Hadoop JobTracker, IBM Innovation centre.

[9] Patrcia A. Oliveira , Jos Marcos Nogueira, Germn Goldszmidt (2011). Availability in Telecommunication Management Distributed Systems, IBM Research - Hawthorne, NY, USA.

[10] J. C. Laprie (1984). Dependability evaluation of software systems in operation, IEEE Transaction on software engineering SE-10(6).

[11] Y.S. Dai, M. Xie, K.L. Poh, G.Q. Liu (2002). A study of service reliability and availability for distributed systems, Department of Industrial and Systems Engineering, National University of Singapore, Kent Ridge Crescent, Singapore, Singapore 119 260.

[12] C. D. Lai, M. Xie, K. L. Poh, Y. S. Dai, P. Yang (2002). A model for availability analysis of distributed software/hardware systems, Northern Telecom, Toronto, Canada.

[13] Greenplum MR High-Performance Hadoop (2012). Retrieved 4th May 2012 from www.greenplum.com.

[14] Alysson N. Bessani, Vinicius V. Cogo, Miguel Correia, Pedro Costa, Marcelo Pasin, Fabricio Silva, Luciana Arantes, Olivier Marin, Pierre Sens, Julien Sopena. Making Hadoop MapReduce Byzantine Fault-Tolerant, Universidade de Lisboa, Faculdade de Ciˆencias, LASIGE – Lisboa, Portugal.

[15] Z. Jelinski, P.B. Moranda (1982). Software Reliability Research, in: W. Freiberger (Ed.), Statistical Computer Performance Evaluation, Academic Press, New York.

Authors

Dr. Ekpe Okorafor has over 15 years' experience in large complex computing systems, network architectures and IT solutions. He received M.Sc. and Ph.D. degrees in Electrical and Computer Engineering in 2002 and 2005 respectively, from Texas A&M University. He has worked in leading research labs including IBM Watson Research and Almaden Research Centers and has managed large Information Technology departments. He is also a visiting professor at the Afric University of Science & Technology. Dr. Ekpe Okorafor's expertise includes computer architecture, mobile networks, virtualization, grid/cloud computing, embedded systems, security, performance modelling and optimization. (E-mail: eokorafor@aust.edu.ng)

Mensah Kwabena Patrick is with the Department of Computer Science, Faculty of Mathematical Sciences at the University for Development Studies, Navrongo, Ghana. (E-mail: menkpat@yahoo.com).