Enhancing NameNode Fault Tolerance in Hadoop Distributed File System

Ohnmar Aung
University of Computer Studies Yangon, Myanmar

Thandar Thein
University of Computer Studies, Yangon, Myanmar

ABSTRACT
In today's cloud computing environment, Hadoop is applied for handling huge data, tens of terabytes to petabytes, with commodity hardware (HDFS) for storage and software (MapReduce) for parallel data processing. In Hadoop version 1.0.3, there is a single metadata server called NameNode which stores the entire file system metadata in main memory and most of I/O operations are associated with those credential metadata. Hadoop is out of commission if NameNode is crashed because it works on memory which becomes exhausted due to multiple concurrent accesses [3]. Therefore, NameNode is a single point of failure (SPOF) in Hadoop and it has to tolerate faults. To solve this issue, a proactive predictive solution is proposed for enhancing NameNode fault tolerance. The solution is designed to proactively calculate the predicted time to crash of NameNode due to resource exhaustion by evaluating the use of powerful Back Propagation Algorithm Neural Network. The proposed approach can give prediction accuracy with minimal error compared to the actual result. Therefore, NameNode's single point of failure can overcome through proposed proactively predicting the time to crash of NameNode caused by memory resource exhaustion.

Keywords
HDFS, NameNode, Memory Resource Exhaustion Prediction, Back Propagation Neural Network

1. INTRODUCTION
In the world of cloud computing, companies and organizations have a tremendous amount of data (big data) that needs to be analyzed and processed very quickly. Therefore, companies like IBM, Amazon, Yahoo!, etc apply Hadoop to big data analysis since it is capable of handling huge data, tens of terabytes to petabytes, with commodity hardware and software. Hadoop is designed as master worker architecture. Master nodes in Hadoop are responsible for handling data storage processing whereas worker nodes are only used for storage purpose. Hadoop separates storage server for data and metadata and all information associated with the entire cluster can only be retrieved via a single metadata server, called NameNode.

NameNode is the heart of Hadoop for keeping the whole file system information. There is only one single NameNode per the cluster in Hadoop release series 1.0.3. Most of I/O operations in the cluster are mainly associated with metadata and NameNode does its entire house holding in memory. When multiple concurrent processes come, NameNode is not only busy but also start to starve resource exhaustion. If the machine running the NameNode were obliterated, all the files on the file system would be lost since there would be no way of knowing how to reconstruct the files [13]. NameNode's failure makes the whole cluster unavailable. Therefore, NameNode has to tolerate fault so that Hadoop can continue giving services to its users.

In the proposed system, a proactively predictive model for NameNode memory exhaustion is presented. Proactive approach takes preventive actions using monitoring tools that can predict anticipated faults before failures [14]. The proposed system is evaluated proactive predictive solution using the system metrics. Therefore, a monitoring agent is applied to collect the required system metrics. The proposed proactive predictive model predicts NameNode failure by calculating the predicted time to crash using back propagation algorithm. The experimental results show that the prediction achieves accuracy of 99.9 % on average. By applying the proposed model in Hadoop version 1.0.3, the level of fault tolerant in NameNode can be enhanced in terms of prediction accuracy which dictates minimal error in the predicted value.

In this paper, the preliminary concept related with the proposed system is presented in Section 2. Then Section 3 highlights issues and contributions in the current cloud infrastructures which are based on Hadoop. The proposed framework is explained in Section 4. The model generation used for the proposed system is described in Section 5 which mentions with experimental preparation and results. The whole presentation is concluded in Section 6.

2. PRELIMINARY CONCEPTS
Due to having the ability of handling massive volumes of data, Hadoop is everywhere in today's cloud platform. However, designing a single metadata server (NameNode) in Hadoop version 1.0.3 limits the system services. NameNode stores the whole file system metadata in memory. It manages queries by clients to carry out standard file system operations

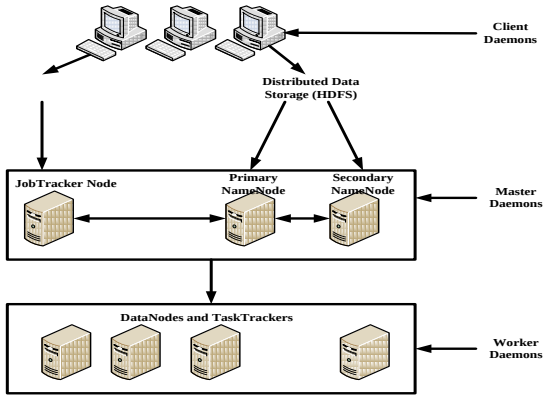


Figure 1: Single Point of Failure in Hadoop

such as add, copy, move or delete files and does all of its house holding in memory. Therefore, memory in NameNode becomes exhausted due to multiple concurrent accesses and it

is a single point of failure (SPOF) in Hadoop cluster. In the proposed system, NameNode failure is prevented by enhancing its existing fault tolerant with the efficiency of proactively predictive approach which uses back propagation algorithm.

3. PROBLEM STATEMENT

Hadoop is popular for having its storage system (HDFS) and parallel data processing framework (MapReduce). HDFS, the Hadoop Distributed File System, is responsible for storing huge data on the cluster. It is designed to run on commodity hardware and can supports write-once-read-many semantics on files. The basic idea of MapReduce is to partition a large problem into smaller sub problems which are tackled in parallel by separate multiple workers. Final result is produced by summing the output of each sub worker [10]. Providing high throughput access to applications that have large data sets makes HDFS different from traditional file systems. In order to give such significant service, Hadoop is implemented as master-worker architecture. Master nodes take role of data processing whereas worker nodes are used as storages purpose.

In cloud computing environment, Hadoop is widely applied for big data analytic. It can give so many advantages to its consumers; meanwhile, it also has been facing with problems to be solved. Followings are found as issues for Hadoop used in today's cloud infrastructures:

- In cloud computing clusters, there can be failures occurred frequently and this can cost extremely to roll back the system into original states.
- NameNode in Hadoop is still the problem of Single Point of Failure (SPOF).
- Memory resource exhaustion impacts heavily on NameNode to happen failure.
- As cloud computing clusters grow in size, maintaining the health of these clusters becomes increasingly challenging because those systems can crash at any time.

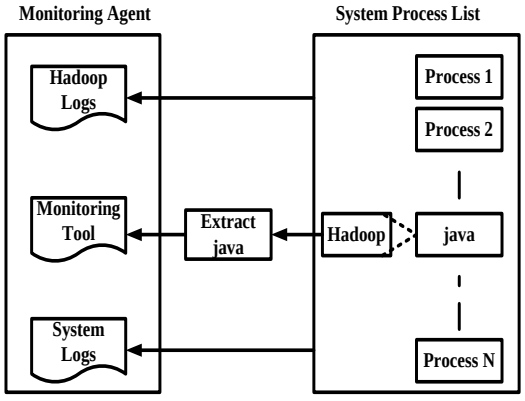


Figure 2: Proposed Proactive Predictive Framework for NameNode Fault Tolerant

Among the above issues, monitoring tools are used to detect potential system failures in order to prevent the cluster from being down [11]. The second problem can be solved by having redundant NameNodes [5]. Resource exhaustion in NameNode can be predicted by calculating the system's time to crash based on collected data via monitoring tool [9]. Final issue is concerned with the system's performance factor: high availability [4]. The proposed system is inspired by exploring above issues. In the later section, process flow of NameNode crash time prediction is presented with theoretical as well as experimental results.

4. PROACTIVE, PREDICTIVE SOLUTION FOR NAMENODE FAULT TOLERANCE

Resource exhaustion impacts host processes running significantly [14]. In Hadoop release series 1.0.3, NameNode is a single point of failure and may primarily be broken down by resource exhaustion. It operates mostly on RAM for instant lookup requests and replies [5]. As long as the parallel connections to the system are increased, the memory consumption of NameNode for each process will be fluctuated in accordance with the nature of the requested process and finally resources become exhausted. Therefore, a proactive predictive solution for enhancing NameNode fault tolerant is presented in the proposed system. The major parts included in the proposed system are discussed in the following sections.

4.1 NameNode Fault Tolerant System Architecture

The proposed system is modeled on Active-Standby NameNode with different configurations. There are three major components in the proposed framework; Active NameNode, Standby NameNode and Monitoring Agent. The health of the Active NameNode is detected with proposed monitoring agent who collects the system traces (memory usage of each process) and then makes proactively predicting the time to crash of the Active NameNode. After that, it informs to the system administrator who is responsible for creating new NameNode. Depending on the situation of threshold violation or not, the administrator makes decision. The Standby NameNode has to take role of the Active NameNode and its activation is dependent on the system administrator's decision. Being known the state of the Active NameNode based on proactively predicting time to crash threshold based mechanism in advance, the administrator has enough time to make decision whether it is required to create new NameNode or not and the proposed proactive predictive NameNode fault tolerance architecture is illustrated in Figure 2.

4.2 NameNode Memory Resource Monitoring

To achieve fault tolerance in NameNode with efficiency of the proposed proactive predictive solution, the required system metrics has to be monitored using certain tools and techniques. So, a monitoring agent is applied to watch the process status of NameNode in the proposed system.

4.2.1 Monitoring Agent

There are many system metrics that can impact system performance and they can be traced via certain monitoring tools such as Nagios[9]. As for the proposed system, in order to predict the time to crash of NameNode, the agent has responsible for capturing memory usage of each process running in the system. So, it collects all the data from one full experiment using a monitoring script. The step by step processes in the monitoring script is shown in Figure 3. The traces monitored via a script are only a raw data set as described in Table 1. These traces cannot be processed directly in further process such as training and testing phase.

Therefore, data enriching process is an essential step for future processing.

4.2.2 Data Enriching

The main job is to add required variables derived from the system metrics monitored [9]. For instance, memory usage for each process may not be constant overtime because NameNode may have to consume in accordance with the process's nature. So, the resource usage per process may be linear or non-linear with respect to the time. Depending upon resource usage, the system downtime may be nearer or further. Generally, the time to crash (TTC) at certain time t for a particular system could then be easily computed by using the following formula:

$$TTC_t = \frac{SAM_i - CUM_{i,t}}{CS_t} \quad (1)$$

Where,

TTC= Time To Crash at time t
SAM = System Available Memory at time t
CUM = Currently Usage Memory at time t
CS = Consumption Speed at time t

In the above equation, the system available memory (SAM) and currently usage memory for resource i at time t (CUM) can be generated in data enriching phase. However, the

1. Set the status = "true"
2. While (true)
   - Get the current system time with "date" command
   - Append and save the time in a text file e.g date >> result.txt
   - Invoke "top" command (to know what processes are currently running and show how much the system resource they consume in percentage)
   - Set loop count n = 1
   - Select batch mode "b"
   - Extract only hadoop process with "grep" command (hadoop appears as "java" in the currently running processes lists).
   - Save and append the extracted record to a result file top –b –n 1 | grep java >> result.txt

2.3 Check the status
3. Go to step 2
4. Exit

Figure 3: Processes of the monitoring script

consumption speed (CS) for each process may not be same over time. In Hadoop, there are many factors influencing the resource consumed by each process [8]. MapReduce job type spends memory nearly at constant rate since it does map and reduce task in accordance with the size of the input split

Table 1: Sample Collected Data

Max Memory (MB)	Start Time	End Time	Usage Memory (%)
512	10:52:18	10:52:23	8.32
512	10:52:23	10:52:28	10.13
512	10:52:28	10:52:33	13.23
512	10:52:33	10:52:38	13.5
512	10:52:38	10:52:43	13.3
512	10:52:43	10:52:48	13.35
512	10:52:48	10:52:53	12.07

Table 2: Enriched Data

Max Memory (MB)	Start Time	End Time	Usage Memory (MB)	Speed (MB per second)
512	10:52:18	10:52:23	42.5984	0
512	10:52:23	10:52:28	51.8656	9.2672
512	10:52:28	10:52:33	67.7376	15.872
512	10:52:33	10:52:38	69.12	1.3824
512	10:52:38	10:52:43	68.096	-1.024
512	10:52:43	10:52:48	68.352	0.256
512	10:52:48	10:52:53	61.7984	-6.5536

defined on the input file [6]. In contrast, multiple concurrent uploading and downloading to the same or different files in Hadoop may be different from MapReduce in resource usage. It may take sometimes much memory than the former job type. Therefore, consumption speed for each process is calculated using EWMA (Exponential Weighted Moving Average) [8].

Exponentially weighted moving average (EWMA) is a type of infinite impulse response filter that applies weighting factors which decrease exponentially. The weighting for each older datum decreases exponentially, never reaching zero [8].

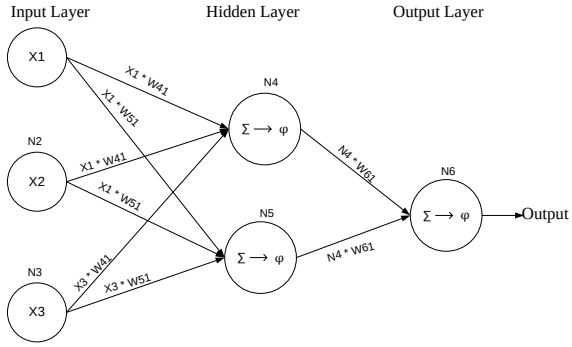


Figure 4: A Three Layer of Artificial Neural Network

In the proposed system, the EWMA for a series Y (data samples: 100) may be calculated recursively as follow:

$$S_n = \propto Y_n + (1 - \propto) S_{n-1} \quad (2)$$

$$\propto (t_n - t_{n-1}) = 1 - \exp \frac{t_n - t_{n-1}}{W * 60} \quad (3)$$

Where,

S_n = value of EWMA at any time period n
Y_n= memory usage at time period n
t_n- t_{n-1} = time difference
α = coefficient (a constant smoothing factor between 0 and 1)
W = process execution time

There are many ways to initialize S1, however, most commonly by setting S1 to Y1 (also same in the proposed system). It depends on the applications used and the required accuracy of the result. "W" means the period of time (process execution time) in minutes and it is set "15 minutes" for the proposed system since it is assumed that each job will be submitted 15 minutes after another. In Hadoop, each node (data nodes as well as job tracker) reports their status to NameNode every 3 seconds to ensure that they are alive. Based on this fact, the time difference between resource usages of each process is limited to 5 seconds in the proposed system. Finally, the whole collected system traces are enriched and the results are as shown in Table 2.

4.3 NameNode Memory Resource Exhaustion Prediction

Artificial neural network is widely used in many applications such as pattern classification, function approximation, clustering and prediction etc.) [1]. Multilayer perceptron learning is a type of neural network and has been applied to solve some difficult and diverse problems by training the network in a supervised manner with its popular algorithm called back propagation algorithm [12]. The proposed system is intended to build a model that can proactively predict the system time to crash by training the perceptrons of back propagation algorithm.

Basically, a back propagation algorithm has two types of pass: forward and backward. The synaptic weights of the networks are fixed in the forward pass. By contrast, during the backward pass, the synaptic weights are all adjusted in accordance with an error correction rule. Updating synapse weights makes the network output move closer to the desired responses, and finally produces minimal error [11]. A sample of three layer neural network is shown in Figure 4.

Neuron is the basic element of the network and is also called computation nodes (N1, N2… N6). The network has generally two layers: 1) input layer in which each neuron accepts input: $y_i(n)$(X1, X2, X3) adds product of some initial weight values:$w_{ji}(n)$(W41, W42) (range is between 0 and 1) and produces the partial output and 2) output layer where the partial result: $v_j(n)$ produced by the former layer is passed through nonlinear function called neuron activation function: $\varphi(v_j(n))$ and finally produces the predicted output: $y_j(n)$by equation 5. The error signal: $e_j(n)$ is calculated by subtracting the actual network response from the desired output using equation 6. The prediction accuracy of the network can know by looking at the error signal. Sometimes, the network cannot predict the result using these two layers and needs extra processing layers called, hidden layers [10]. Depending upon the output produced by the network at first time, one or more hidden layers are required. There are important parameters such as learning rate and local gradient which has to be adjusted so that the network can predict the desired output quickly as well as accurately.

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \quad (4)$$

$$y_j(n) = \varphi\left(v_j(n)\right) \quad (5)$$

$$e_j(n) = d_j(n) - y_j(n) \quad (6)$$

On account of predicting with back propagation algorithm for the proposed system, four attributes are fed into the network. Three out of four are used as inputs to the network such as available memory, currently usage memory, consumption speed and the rest is the time to crash (the desired response). In order to achieve the desired output, the network was trained iteratively. For each loop, the weight coefficients of the nodes in each layer are modified using error correction rule: $\Delta w_{ji}(n)$ in which gradient descent: $\delta_j(n)$ and learning rate: η were adjusted for seeking a direction for weight change the value of the error signal.

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \quad (7)$$

$$\delta_j(n) = e_j(n) \dot{\phi}_j(v_j(n)) \quad (8)$$

What's more, the data set is divided into training and testing. 10 fold cross validation is used. Initially, the network is trained with the training data set. When the error signal reaches nearly zero, i.e., the network output is very close to the desired output, stops the network and captures the weight value because this parameter is the core of the network when estimating the future output. Then, the testing data set is given to the network whether it is able to work also on data that was not used in the previous process.

5. SYSTEM EVALUATION

5.1 Experimental Setup

To evaluate the effective of the proposed approach, an experimental environment is set up with commodity hardware and necessary software installation is done as shown in Table 3. A machine having specifications with Memory 4GB, Hard Disk 1 TB, CPU Core i3 and Window 7 version is chosen as a host. On that machine, three virtual machines are created using VMware Workstation 9 with equal specifications such as Memory 512 MB, Hard Disk 20 GB, and CPU Core i3. Each of them is connected using 100 Mps and Ubuntu 10.04 Lts is used as supporting OS as shown in Table 3.

Table 3: Test Bed Specifications

	Host	VM1	VM2	VM3
Memory	4GB DDR3	512MB DDR3	512MB DDR3	512MB DDR3
Hard Disk	1 TB	20 GB	20 GB	20 GB
CPU	Core i3	Core i3	Core i3	Core i3
OS	Window 7	Ubuntu 10.04 LTs Kenrel 2.6.32	Ubuntu 10.04 LTs Kenrel 2.6.32	Ubuntu 10.04 LTs Kenrel 2.6.32
Hadoop Version		1.0.3	1.0.3	1.0.3
VMware Workstation		9	9	9
Network (Mbps)	100	100	100	100

5.2 Analysis of Resource Usage Behavior

Selecting hardware that provides the best choice of performance and economy for a given workload requires testing and validation. To show the strength of proposed proactive predictive framework for prediction of time to crash in NameNode due to resource exhaustion, analysis of resource usage is done by releasing processes and monitoring how process consume resource to complete submitted jobs.

In the experimental evaluation of the proposed system, it was found that resource behavior was quietly different from one job type to another. It is solely dependent upon the characteristic of the running processes. For instance, when writing Java programs, the memory requirement for a single loop may be constant almost every time. However, for a program having recursive condition, it consumes memory at a gradually increasing rate and later this progressive consumption makes the underlying system memory resource face with exhaustion [8]. In this section, the root cause of random resource consumption is discussed through two different job type running analysis in Hadoop. The first part of the analysis was made on running MapReduce jobs. And, the second part was emphasized on working with random job types: uploading and downloading files. Finally, deduction was made on the two analysis results.

5.2.1 Running Map Reduce Job in Hadoop

In this test set, when a simple WordCount program with a TXT file having size of 100 MB was submitted, memory usage in NameNode grows up initially. By nature of MapReduce, the program usually reserves required amount of heap space to run its tasks (map tasks and reduce tasks). For instance, default heap space is set with 256 MB and it can change depending on the process requirement. The changes can be made in mapred-site.xml configuration file of Hadoop with the command "-Xmx256m" [13]. When the program detects that it has enough space to run the task, it starts to work and consumes memory. During the map and reduce task for a given job, the resource consumption is at regular rate. Finally, it starts to release spaces when the user submitted job is completed successfully. The resource usage of running WordCount for a TXT file with 100MB is shown in Table 4. It can be found that the memory usage for running simple WordCount of MapReduce is in steady state.

5.2.2 Running Simple I/O Job in Hadoop

As the second test on surveying resource consumption of the processes in Hadoop, simple job of uploading and downloading with multiple file types having different sizes is submitted. When there is single user who uploads and downloads files, the resource usage is nearly regular rate. However, when multiple concurrent users are allowed to submit jobs, the memory consumption speed becomes to increase and later, within a few seconds (around 5 seconds), the speed approached to fluctuate. As long as there is increasing number of users submitting jobs (upload, download) concurrently, the behavior of the resource usage is gradually changed into abnormal as shown in Table 5.

5.2.3 Comparison of Resource Usage Difference between Two Job Types

Figure 5 shows the result of consumption speed by two types of jobs running in Hadoop. The red line represents the resource usage in running with parallel upload and downloads files whereas the blue line expresses the memory consumption speed of running simple wordcount MapReduce process.

Table 4: Memory Usage in Running MapReduce Job

Max Memory (MB)	Usage Memory (MB)	Speed (MB per second)
512	65.792	1.16634
512	65.792	0
512	65.792	0
512	65.792	0
512	65.792	0
512	65.792	0
512	52.457	-13.335

Table 5: Memory Usage in Running Simple I/O Job

Max Memory (MB)	Usage Memory (MB)	Speed (MB per second)
512	78.08	0
512	66.8585	-11.2215
512	73.5575	6.69901
512	73.1023	-0.45517
512	72.9175	-0.18483
512	73.0455	0.128

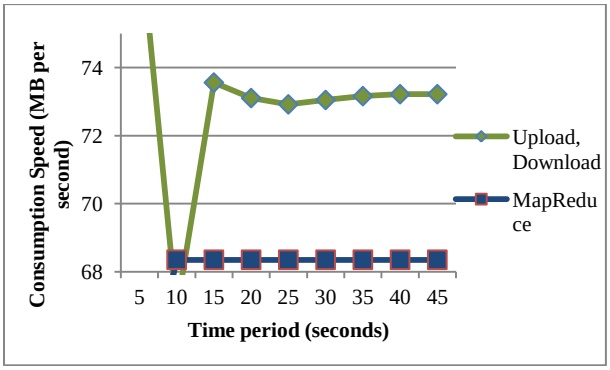


Figure 5: Resource Usage Difference between Two Job Types

In the case of running wordcount program, there is no significantly difference in resource consumption. And, the speed is nearly constant over time. Alternatively, the result of second testing is rather different. In this run, there is a fluctuation in resource usage. The reason of this result is because the application clients are making a continuous downloading and uploading files in the system and this result on a significant overhead of memory consumption.

5.3 Proactively Predicting Time To Crash with Back Propagation Algorithm of Neural Network

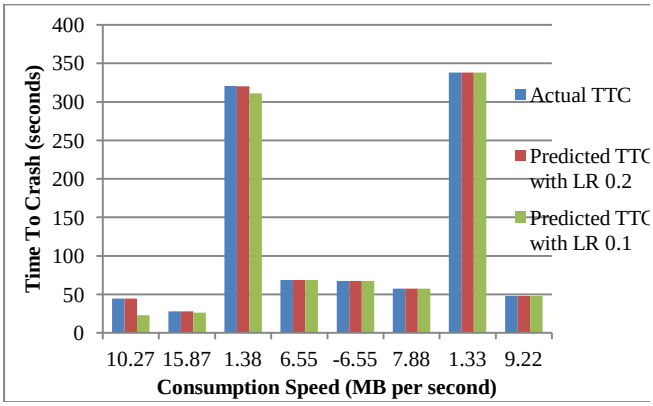
In this section, the back propagation algorithm of neural network is applied to predict the time to crash of NameNode based on the collected traces.

Table 6: Training Result with Learning Rate 0.1

Max Memory (MB)	Usage Memory (MB)	Speed (MB per second)	Actual TTC (seconds)	Predicted TTC (seconds)
512	42.5984	0	0	0
512	51.8656	10.2672	44.81596	44.78924
512	67.7376	15.872	27.99032	27.97372
512	69.12	1.3824	320.3704	320.181
512	68.096	-1.024	-433.5	-433.253
512	68.352	0.256	1733	1731.963
512	61.7984	6.5536	68.69531	68.6619

Table 7: Training Result with Learning Rate 0.2

Max Memory (MB)	Usage Memory (MB)	Speed (MB per second)	Actual TTC (seconds)	Predicted TTC (seconds)
512	42.5984	0	0	0
512	51.8656	10.2672	44.81596	44.80331
512	67.7376	15.872	27.99032	27.98246
512	69.12	1.3824	320.3704	320.2807
512	68.096	-1.024	-433.5	-433.383
512	68.352	0.256	1733	1732.509
512	61.7984	6.5536	68.69531	68.67933



TTC = Time To Crash
LR = Learning Rate

Figure 6: Comparison of Actual TTC and Predicted TTC using two learning rates

The traces generated by enriching process are used as data set for the proposed framework. Each record set is created having the number of process per each sample period (5 seconds in this case). As for using back propagation algorithm , the one-hundred (100) samples were separated into data set that can be used for training network and testing data set which is used to test whether the network can produce the desired output when unknown input was given. In the proposed system, the network is trained with two learning rate parameters so that the network can produce the predicted output which is nearly matched with the actual output.

Firstly, the network is trained with learning rate 0.1 to predict the result and the output is recorded. The result achieved is as shown in Table 6. In the second test set, the network is set up again with learning rate (0.2) to train the total samples and stops until the output approaches to the target. As before, the rest of the data set is pushed into the network and the result is regarded and the comparison results between the actual and predicted one is as illustrated in Table 7.

5.4 Result Discussion

It can be found that the predicted result obtained through training with learning rate 0.2 (red vertical bar) as shown in Figure 6 is closer to the actual result (blue vertical bar)than testing with learning rate 0.1 which is described as the green vertical bar. It may be clearer to know which of these two learning rates the best predictor to use is. Therefore, Mean Absolute Error (MAE) for the two network output (learning rate 0.1 and 0.2) is computed.

The MAE formula is expressed below:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left(d - \hat{d}\right)^2 \quad (9)$$

Where, d = Actual Time To Crash Value
$\hat{d}$ = Predicted Time To Crash Value
N = Number of Samples

In the proposed system, the value of MAE is calculated for both of output produced by two learning rates. The MAE value using learning rate 0.2 is smaller than those of MAE for learning rate 0.1 as mentioned in Table 8. The reason is that the network trained with learning rate 0.1 produces the predicted output: 22.8560 whereas the actual result is 44.8159. And, this makes the error value to be significantly high in comparison with the predicted result: 44.8033 obtained by using learning rate 0.2. Besides, it took over 2000 epochs (number of training cycles) for the network to predict the targeted result due to learning rate 0.1 whereas training time only lasted nearly 1000 epochs for the network when using learning rate 0.2.

6. RELATED WORK

Among many failures found in Hadoop, resource exhaustion was the most common problems. Since NameNode in Hadoop is still single point of failure, it can be an interrupt in getting high availability of the system. Many researchers have been done using failure detection and prediction so that Hadoop can overcome issues with NameNode. Feng Wang and his colleges [7] used metadata replication scheme to enhance Hadoop NameNode High Availability. A different approach is presented by Cristina L. Abad and his buddies [2]. They introduced a synthetic workload generator called “Mimesis” and evaluated its usefulness through a case study in a least recently used metadata cache for the Hadoop Distributed File System.

Table 8: Comparison of MAE value for Two Learning Rates

Actual TTC (d)	LR(0.2) d1	LR(0.1) d2	$(d-d1)^2$	$(d-d2)^2$
44.81596	44.8033149	22.85608	0.00016	482.2363
27.99032	27.98246131	26.33111	6.18E-05	2.752971
320.3704	320.2807235	310.9505	0.008042	88.73382
68.69531	68.67932591	68.65101	0.000255	0.001962
67.6953	67.67964826	67.6523	0.000245	0.001849
57.63399	57.62074645	57.59769	0.000175	0.001318
338.1538	338.0765	337.9423	0.005975	0.044729
47.84444	47.83383492	47.81575	0.000112	0.000823
MAE			0.00187	71.7217

Similar works can be found, however, as for concerning with web server crash, a framework for predicting of time to failure due to consuming resource (memory) randomly using M5P (a Decision Tree Algorithm). To prove the availability of the system, the authors illustrated with random injection of memory leaks to the proposed system [9] and recorded the logs and finally gave predicted time to failure to the administrator. Dhurba Borthakur, project lead of Apache Hadoop Distributed File System creates Avator Node to overcome the problem of single point of failure happened in NameNode of Hadoop [5]. To address the problem of NameNode, the authors in Intel Distribution for Apache Hadoop software uses shared edits directory with two ways to grant accesses: Network File Share (NFS) and Quorum Journal Manager.

All of the above researches are intended to raise system performance as well as give fault tolerance based on prediction approach. Like those, the proposed system also presents resource exhaustion prediction approach. But, using a proactive predictive solution for enhancing NameNode fault tolerance in Hadoop release series 1.0.3 makes the proposed system differ from the earlier systems. In addition, the system can make prediction accuracy up to 99.9% compared to the actual result.

7. CONCLUSION

Resource exhaustion in NameNode of Hadoop version 1.0.3 makes the cluster unavailable. In the proposed system, a monitoring agent proactively collects the system metrics and the traces are applied in time to crash calculation by using back propagation algorithm. The predicted results trained by the algorithm shows in terms of accuracy with minimal error. By viewing the predicted time to crash of NameNode, the system administrator can make decision whether it is time to create new NameNode or not. Therefore, the proposed proactive predictive solution can be utilized in preventing NameNode failure.

8. REFERENCES

[1] Anil K. Jain, "Artificial Neural Networks: A Tutorial", in Proceedings of Neural Computing: Companion issue to Engineering, Vol. 29 Issue 3, March 1996, pp. 31-44

[2] Cristina L. Abad, Huong Luu, Nathan Roberts, Kihwal Lee, Yi Lu and Roy H. Campbell, "Metadata Traces and Workload Models for Evaluating Big Storage Systems", in Proceedings of IEEE 5th International Conference on Utility and Cloud Computing (UCC), Chicago, IL, November 5-8, 2012, pp. 125-132.

[3] Chuck Lam, "Hadoop in Action", Manning Publications Co. 180 Broad St. Suite 1323, Stamfor, CT 06901, December 22, 2010.

[4] Diane Hatcher, "Considerations for Implementing a Highly Available or Disaster Recovery Environment," SAS Institute Inc, Cary, NC, USA, 2011.

[5] Dhruba Borthakur, "Apache Hadoop and Its Usage in Facebook", UC Berkeley, April 4, 2011. Online Available : http://www.facebook.com/hadoopfs

[6] Eric Sammer, "Hadoop Operations", O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, United States of America, September 9, 2012.

[7] Feng Wang, Jie Qiu, Jie Yang, "Hadoop High Availability through Metadata Replication", IBM Research, China, 2009.

[8] Javier Alonso and Jordi Torres, "Predicting Web Server Crashes: A Case Study in Comparing Prediction Algorithms", in Proceedings of 5th IEEE International Conference on Autonomic and Autonomous Systems (ICAS'09), Valencia, April 20-25, 2009, pp. 264-269.

[9] Javier Alonso Lopez, "Proactive Software Rejuvenation Solution for Web Environment on Virtualized Platforms," Doctoral thesis, Barcelona, Spain 2011.

[10] Jimmy Lin and Chris Dyer, "Data-Intensive Text Processing with MapReduce", University of Maryland, College Park, April 11, 2010.

[11] Roman Dudko, Abhishek Sharma, Jon Tedsco, "Effective Failure Prediction in Hadoop Clusters", March, 2012. Online Available: http://www.techrepublic.com/resource-library/whitepapers/effective-failure-prediction-in-hadoop-clusters/

[12] Simon Haykin, "Neural Network: A Comprehensive Foundation," Prentice Hall, Delhi, India, 1999.

[13] Tom White, "Hadoop: The Definitive Guide", O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, United States of America, May 2012.

[14] Xiaojuan Ren, Seyong Lee, Rudolf Eigenmann, Saurabh Bagchi, "Prediction of Resource Availability in Fine-Grained Cycle Sharing Systems Empirical Evaluation", J Grid Computing (2007), Vol 5, pp 173-195.