Analyzing and Predicting Failure in Hadoop Clusters Using Distributed Hidden Markov Model

Conference Paper · June 2017

DOI: 10.1007/978-3-319-28430-9_18

Bikash Agrawal
University of Stavanger (UiS)
12 PUBLICATIONS **86** CITATIONS
SEE PROFILE

Chunming Rong
University of Stavanger (UiS)
324 PUBLICATIONS **5,184** CITATIONS
SEE PROFILE

Analyzing and Predicting Failure in Hadoop Clusters using Distributed Hidden Markov Model

Bikash Agrawal, Tomasz Wiktorski, Chunming Rong

Department of Computer and Electrical Engineering
University of Stavanger
Stavanger, Norway
{bikash.agrawal,tomasz.wiktorski,chunming.rong}@uis.no

Abstract In this paper, we propose a novel approach to analyze and predict failures in Hadoop cluster. We enumerate several key challenges that hinder failure prediction in such systems: heterogeneity of the system, hidden complexity, time limitation and scalability. At first, clustering approach is applied to group similar error sequences, which makes training of the model effectual subsequently Hidden Markov Models (HMMs) is used to predict failure, using the MapReduce programming framework. The effectiveness of the failure prediction algorithm is measured by precision, recall and accuracy metrics. Our algorithm can predict failure with an accuracy of 91% with 2 days in advance using 87% of data as training sets. Although the model presented in this paper focuses on Hadoop clusters, the model can be generalized in other cloud computing frameworks as well.

Keywords: Hadoop; Failure prediction; Hidden Markov Model; Failure analysis, Machine Learning

1 Introduction

The cluster system is quite commonly used for high performance in cloud computing. As cloud computing clusters grow in size, failure detection and prediction become increasingly challenging [18]. The root causes of failure in such a large system can be due to the software, hardware, operations, power failure and infrastructure that support software distribution [24]. The cluster system dealing with a massive amount of data needs to be monitored and maintained efficiently and economically. There have been many relevant studies on predicting hardware failures in general cloud systems, but few on predicting failures in cloud computing frameworks such as Hadoop [32]. Hadoop is an open-source framework for distributed storage and data-intensive processing, first developed by Yahoo. Hadoop provides an extremely reliable, fault-tolerant, consistent, efficient and cost-effective way of storing a large amount of data. Failure in storing and reading data from the large cluster is difficult to detect by human eyes. All the events and activities are logged into their respective application log files. Logs provide information about performance issues, application functions, intrusion, attack attempts, failures, etc. Most of the applications maintain their own logs. Similarly, HDFS system consists of DataNode and NameNode logs. The logs produced by NameNode, secondary NameNode and DataNode have their individual format and content.

The prime objective of the Hadoop cluster is to maximize the job processing performance using data-intensive computing. Hadoop cluster normally consists of several nodes and can execute many tasks concurrently. The job performance is determined by the job execution time. Execution time of a job is an important metric for analyzing the performance of job processing in the Hadoop cluster [4]. As Hadoop is a fault-tolerant system if the nodes fail, then the node is removed from the cluster in the middle of the execution and the failed tasks are re-executed on other active nodes. However, this assumption is not realistic because the master node can crash. Many researchers reported that the master node crash is a single point of failure and needs to be handled [19] [31]. Even the failure of the data node results in higher job execution time, as the job needs to re-execute in another node. Failure nodes are removed from the cluster so that the performance of the cluster improves. Prediction methods operate mostly on continuously available measures, such as memory utilization, logging or workload, to identify error pattern. Our analysis in this paper is mostly only on time of occurrence of different types of error events that ultimately cause failure. This might also help in root cause analysis [34] for automatic triggering of preventive actions against failures.

We used Hidden Markov Models (HMMs) [3] to learn the characteristics of log messages and use them to predict failures. HMMs have been successfully used in speech, handwriting, gesture recognition, and as well also in some machine failure prediction. HMM is well suited to our approach as we have observations of the error messages, but no knowledge about the failure of the system, which is "hidden". Our model is based on a stochastic process with a failure probability of the previous state. As faults are unknown and cannot be measured, they produce error messages on their detection (i.e. present in log files).

Our prediction model is divided into four main parts; First, identifying error sequences and differentiating types of error from the log files. Second, using the clustering algorithms [11] like K-means [16]. Third, training the model. Given the labeled training data, HMM method is used to evaluate maximum likelihood sequence that is used to update the parameters of our model. Last, predicting failure of the system based on the observation of an error sequences. The main idea of our approach is to predict failures by analyzing the pattern of error events that imitates failure. Experimental results for this method can predict failure with 91% accuracy for 2 days in advance (prediction time). It also shows that our approach can compute on the massive amount of datasets. Ultimately, our approach can be used to improve the performance and reliability of the Hadoop cluster.

1.1 Related work

A significant number of studies have been done on the performance evaluation and failure diagnosis of systems using log analysis. However, most of the prediction methods focus only on the system logs, but not on the application logs. Many more studies have been done on predicting hardware failure in the cluster. For example, studies in [27], [24] and [21] provide a proactive method of predicting failure in the cluster, based on system logs. These methods provide failure in hardware level but fail to provide failure of a node in the Hadoop clusters.

Konwinski et al. [17] used X-trace [12] to monitor and improve Hadoop jobs. X-trace allows path-based tracing on Hadoop nodes. Additionally, using X-trace in conjunction with other machine-learning algorithm, they were able to detect failure with high accuracy. Similarly, SALSA [28] is another tool in which system logs are analyzed to detect failure using distributed comparison algorithm. Also, it also shows information on how a single node failure can affect the overall performance of the Hadoop cluster. All of these papers present failure detection algorithm in the Hadoop cluster but lacks prediction algorithm.

Fulp et al. [13] demonstrated that failure prediction in the hard disk using SVMs (Support Vector Machines) with an accuracy of 73% with two days in advance. On the other hand, Liang et al. [18] uses RAS event logs to predict failure in IBM BlueGene/L. They compare their results with Support Vector Machines (SVMs), a traditional Nearest Neighbor method, a customized Nearest Neighbor method and a rule-based classifier, and found that all were out-performed by the customized Nearest Neighbor method. However, these all provide failure prediction algorithm in the different areas, but still lacks the good accuracy of the model.

Hidden Markov Models have been used in pattern recognition tasks such as handwriting detection [22], gene sequence analysis [9] [5], gesture recognition [33], language processing [15] [23], hard drive failure [29] or machine failure [27]. HMM is a widely used model due to it's flexibility, simplicity, and adaptivity. Indeed, as mentioned earlier, Hadoop log data have challenging characteristics, which thus require expert knowledge to transform data into an appropriate form.

1.2 Our Contribution

We proposed a novel algorithm for failure prediction algorithm using MapReduce programming framework, thus achieving better scalability and better failure prediction probability. The proposed model is based on distributed HMM through MapReduce framework in a cloud-computing environment. Through this paper, we also present our idea to increase the performance of the Hadoop Cluster by predicting failure. The accuracy of our model is evaluated using performance metrics (precision, recall, F-measure).

1.3 Paper Structure

Section II gives an overview of the background. Section III introduces the design and approach of our analysis. Section IV evaluates our algorithm and presents the results. Section V concludes the paper.

2 BACKGROUND

2.1 Hadoop

Hadoop [32] is an open-source framework for distributed storage and data-intensive processing, first developed by Yahoo. It has two core projects: Hadoop Distributed File System (HDFS) and MapReduce [7] programming model. HDFS is a distributed file system that splits and stores data on nodes throughout a cluster, with a number of replicas. It provides an extremely reliable, fault-tolerant, consistent, efficient and cost effective way to store a large amount of data. The MapReduce model consists of two key functions: Mapper and Reducer. The Mapper processes input data splits in parallel through different map tasks and sends sorted, shuffled outputs to the Reducers that in turn groups and processes them using reduce tasks for each group.

2.2 Hidden Markov Models

HMM [2] is based on Markov Models in which one does not know anything about observation sequences. The numbers of states, the transition probabilities, and from which state an observation is generated are all unknown. It consists of unobserved states. And each state is not directly visible, but output and dependent on the state is visible. HMM typically used to solve three types of problem: detection or diagnostic problem, decoding problem and learning problem. Using forward-backward algorithm [14] solves diagnostic problem. Using Viterbi algorithm [30] solves decoding problems. And using Baum-Welch algorithm [8] solves learning problem.

3 Approach

In this section, we describe how the useful information from different logs are extracted and the use of HMMs to predict failure from those log files.

The proposed method deals with all the log files associated with Hadoop cluster (HDFS): DataNode and NameNode logs. The log files are collected from the different nodes associated with the cluster. The logs generated from 11-node clusters are stored in HDFS system using Apache Flume collector [1]. The log files contain all unwanted and wanted information that makes it difficult for the human to read. For this reason, pre-processing of logs is needed before storing to HDFS system. In the pre-processing steps, all the log messages are extracted and unwanted and noisy messages are removed. The stored data is further analyzed using HMM model. Failure prediction algorithm is used to detect failure and ignore defective node before running any task.

HDFS system consists of NameNode and DataNode. NameNode is the master node on which job tracker runs. It consists of the metadata (information about data blocks stored in DataNodes - the location, size of the file, etc.). It maintains and manages the data blocks, which are present on the DataNodes. The DataNode is a place where actual data is stored. The DataNode runs three main types of daemon: Read Block, Write Block, and Write-Replicated Block. NameNode and DataNode maintain their own logging format. Each node records events/activities related to reading, writing and replication of HDFS data blocks. Each log is stored on the local file-system of the executing daemon. Our analysis is based on some important insights about the information available through Hadoop logs. Block ID in DataNode log provides a unique identifier, which represents the HDFS blocks that consist of raw bytes of the data file.

Before using log messages to build the model, we structured and appended all the log files into systematized forms. Four steps are involved in our approach; pre-processing, clustering, training, and predicting as shown in figure 1. In first steps, all useful information, such as timestamp, error status, error type, node ID and user ID, are extracted and new log template is created. Since different logs reside on the local disk in different nodes, it is necessary to collect and attach all the log information into a one-log template. In second steps, we use the clustering algorithm to differentiate various types of errors. With the clustering technique, real error types that propagate to failure are recognized. And the third and fourth steps, is the training and prediction algorithm using HMM model, which is discussed in detail below.

We adopted Hidden Markov models (HMMs) for this approach. HMM applies machine-learning techniques to identify whether a given sequence of the message is failure-prone or not. HMM models parameters can be adjusted to provide better prediction. The sequences of an error event are fed into the model as an input. Each error event consists of a timestamp, error ID and error type, which determine the types of error. Failure and non-failure information are extracted from error sequences to create a transition matrix. HMM is characterized by the following modules: hidden states $X = \{x_1, x_2, x_3\}$, observations state $Y = \{y_1, y_2, y_3\}$, transition probabilities A $= a_{ij} = \{P[q_{t+1} = x_j | q_t = x_j]\}$ and emission probabilities B $= b_{ij}$. HMM (λ) is denoted as

$$\lambda = (\pi, A, B) \tag{1}$$

Where, A is the transition matrix whose elements give the probability of transitioning from one state to another, B is the emission matrix giving $b_j(Y_t)$ the probability of observing Y_t. π is initial state transition probability matrix.

The observation symbols $O_1 = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ are referred to error events of the system, and failures are represented as hidden state of HMM as shown in figure 2. Error patterns are used as training set for HMM model if the model transits to a failure state each time a failure occurs in the training data. Two steps are necessary for obtaining training sequences for the HMM. The first

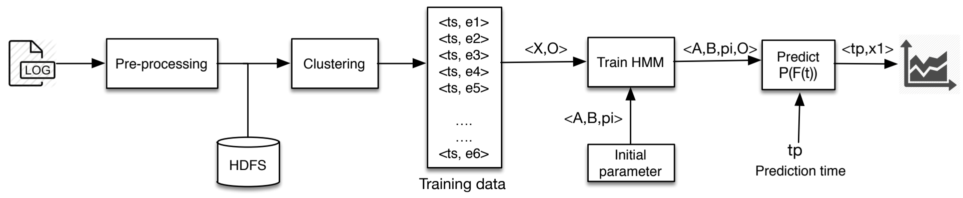


Figure 1: Workflow of failure prediction.

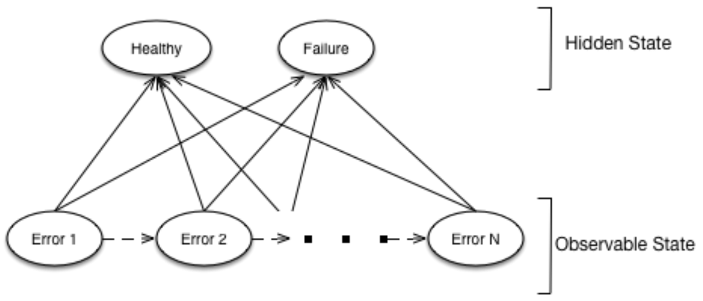


Figure 2: Mapping Failure and errors to Hidden Markov Model.

step involves the transformation of error types into a special error symbol in order to classify different types of error. Error event timestamps and error categories form an *error sequence* (event-driven sequence). HMM applies machine-learning techniques in order to identify characteristic properties of an error sequence. It also helps to detect whether a given error sequence is failure-prone or not. Moreover, we trained the model using past error sequence. The model adjusts its parameters based on those records. The trained model is then applied to the new error sequences to predict failure. Such an approach in machine learning is known as "*supervised learning*". To extract error sequence, timestamp and error ID are extracted in preprocessing step as mentioned earlier. Let "e" represent different types of error in the log files. The series of messages that appear in "e" form a time-series representation of events that occurred. In this paper, all categories of "e" are identify using *k-means clustering technique* [16]. Six different types of error are distinguished from the given log files and the set e would be $(e_1, e_2, e_3, e_4, e_5, e_6)$. This error set is known as *error sequence* or observation for our model.

In the next step, the model is defined using error sequences. Error sequence consists of failure and non-failure information that has occurred within a sliding window of length Δt as shown in figure 3. F is the failure in the system and e_1, e_2, e_3, e_4 represent the error events in the log files. Failure t_p is predicted based on Δt error sequence. HMM models are trained using error sequences. The main goal of training is that failure characteristics are generated from the error sequences. The non-failure model stays rather remains unspecific. Once the

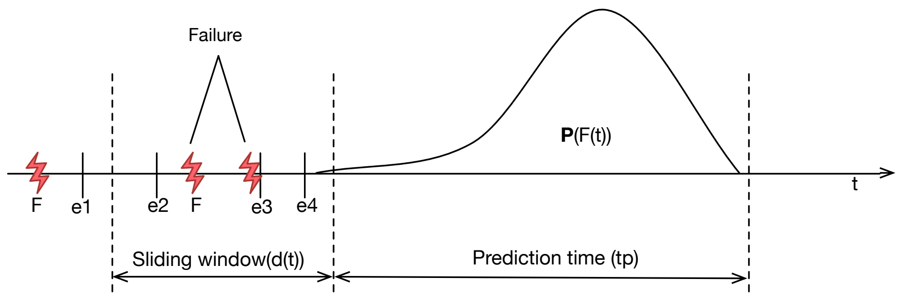


Figure 3: Failure prediction, where e1, e2, e3 and e4 are error sequences, and F is the failure in the system.

models are trained, new upcoming failure is evaluated from the error sequences. To predict upcoming failure, sequence likelihood is computed from HMM models.

In this paper, there is a sequence of log data over timestamp, which we needed to train our HMM model using a set of the sequence of log output as observation O = (info, warn, error, fatal). N= 4 for HMM model is denoting the stages in time that are allowed in different transitions in the HMM training. Each error sequence may result in failure-prone or non-failure system. The Failure-prone system has a similar pattern of errors, which result in failure. The probabilityof log data are computed using Forward-Backward Algorithm as proposed in [8].

Training the model: First, from the log template sequence of an error message is extracted an output sequence composed of 1,2,3,4,5 and 6's, one number for each time step by rounding the timestamp of logs of a job to the nearest integer. Thus, the state transition in the HMM takes place every time step until the absorption state is reached. The choice of time step determines the speed of learning and its accuracy. A log sequence of a job always starts from the state x_1 and ends at x_2, and the initial probabilities for π are fixed to be 0.5. With the output sequence as described, we compute the most likely hidden state transition sequence and the model parameters $\lambda = (A, B, \pi)$.

We have adapted Expectation-Maximization algorithm for training purpose. During training, the HMM parameters π, A, B are optimized. These parameters are maximized in order to maximize sequence likelihood. For initial steps, number of states, number of observation, transition probability and emission probability are pre-specified. In this experiment, initial parameters are calculated from the past observation, such that the model can predict accurately from the initial phase. As training of model progress, the parameter value gets closer to the actual value. Training in HMM is done using Expectation-Maximization algorithm, where backward variable β and forward variable α are evaluated. This algorithm helps to maximize the model parameters based on maximum likelihood. If the model started randomly from a pre-specified HMM parameter, it will take several iterations to get superior parameters, which best fit, the model for prediction. The goal of training datasets is to adjust the HMM parameters such that error sequences are best represented and that the model transits to a failure state each time a failure occurs in the training datasets.

Algorithm 1 Failure State Prediction Algorithm

```
 1: Initialized $O = \{o_1, o_2, ..o_6\}$        ▷ different types of error as observation
 2: $S = \{Healthy, Failure\}$        ▷ two hidden state
 3: $m = 2$        ▷ m is number of hidden state
 4: $n = 6$        ▷ n is number of different types of errors
 5: Initialized $A_{ij}, B_{ij}$        ▷ emission matrix $B_{ij}$ stores the probability of observable
    sequences $o_j$ from state $s_i$  ▷ transition matrix $A_{ij}$ stores the transition probability
    of transiting from state $s_i$ to state $s_j$
 6: Initialized $\Pi$        ▷ an array of initial probabilities
 7: $Y = \{y_1, y_2 ... y_k\}$        ▷ an error sequence of observation
 8: **Map:**
 9: Initialized $StatePathProb$
10: Update $A_{ij}, B_{ij}$
11: $PathProb = StatePathProb * B_{ij}$
12: for each state $s_j$ do
13:     $StatePathProb[j, i] \leftarrow \max_k (StatePathProb[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$
14:     $PathProb[j, i] \leftarrow \arg\max_k (PathProb[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$
15: end for
16: $z_i \leftarrow \arg\max_k (StatePathProb)$
17: $x_i \leftarrow S_i$
18: for $i \leftarrow T-1, ..., 1$ do        ▷ T is length of observable sequence
19:     $z_i \leftarrow PathProb[z_i, i]$
20:     $x_i \leftarrow zi$
21: end for
22: $emit(timestamp, x)$
```

There are a few existing methods such as; Baum-Welch algorithm and gradient-descent techniques, uses iterative procedures to get the locally maximized likelihood. However, this iterative procedure might be significantly slow if the observed sequence is large. In this paper, we proposed a slightly different algorithm to train data, which is significantly faster than the traditional method. The idea is to formulate the probability of the observation sequence O_t, O_{t+1} pairs and then to use Expectation-Maximization algorithm to learn for this model λ.

In order to train the model, there is a need to find the repetitive error sequence in the data. To do so first, we need to compute the likelihood of raw data in the desired range. This problem is computed using EM algorithm. The EM consists of two steps: an expectation (E) step, which creates a function for calculating log-likelihood from the current estimate, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood calculated on the E step. This EM algorithm is carried on a map and reduce task.

After getting all the parameter of the model (λ), the prediction algorithm is used to get all the hidden states that are calculated using maximum likelihood of the past sequences.

Prediction: Failure is the hidden layer in our HMM model. The Hidden state is calculated using Viterbi algorithm. There is a sequence of observations $0 = O_1, O_2....O_n$ with given model $\lambda = (A, B, \pi)$. The aim of Viterbi algorithm is to find optimal state sequence for the underlying Markov chain, and thus, reveal the hidden part of the HMM λ. The final goal of Viterbi is to calculate the sequence of states (i.e $S = \{S_1, S_2, ...S_n\}$), such that

$$S = argmax_s P(S; O, \lambda) \tag{2}$$

Viterbi algorithm returns an optimal state sequence of S. At each step t, the algorithm allow S to retain all optimal paths that finish at the N states. At t+1, the N optimal paths get updated and S continues to grow in this manner. Figure 4 shows details architecture of Viterbi algorithm implementation. The goal is to predict hidden state from the given observation $0 = \{O_1, O_2....O_n\}$. No reducer is used and on each mapper, a local maximum is calculated and state path based on maxima are observed. Two states defined in algorithm 1: healthy and failure are determined based on error sequence.

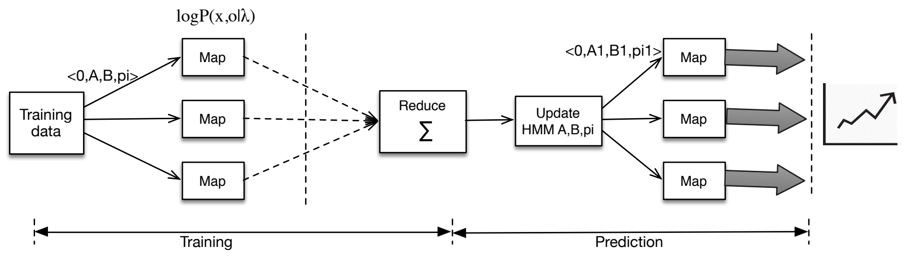


Figure 4: Architecture of an algorithm predicting failure state using MapReduce programming framework.

4 RESULT

Setup: Our cluster is comprised of 11 nodes with CentOS Linux distro, one node each for Namenode, Secondary Namenode, HBase Master, Job Tracker, and Zookeeper. The remaining 6 nodes act as Data Nodes, Regional Severs, and Task Trackers. All nodes have an AMD Opteron(TM) 4180 six-core 2.6GHz processor, 16 GB of ECC DDR-2 RAM, 3x3 TeraBytes secondary storage and HP ProCurve 2650 switch. Experiments were conducted using RHIPE, Hadoop-0.20, Hbase-0.90 Apache releases. Our default HDFS configuration had a block size of 64 MB and the replication factor of 3.

The prediction techniques presented in this paper have been applied to the data generated while performing operations in the Hadoop Cluster. With 1 month of Hadoop log data, we trained HMM model using sliding windows varying from 1 to 2 hours in length. A Large amount of Hadoop log data was generated using SWIM [26], a tool to generate arbitrary Hadoop jobs that emulate the behaviors of true production jobs of Facebook. We used AnarchyApe [6] to create different types of failure scenarios in Hadoop cluster. AnarchyApe is an open-source project, created by Yahoo! developed to inject failures in Hadoop cluster [10].

4.1 Types of error

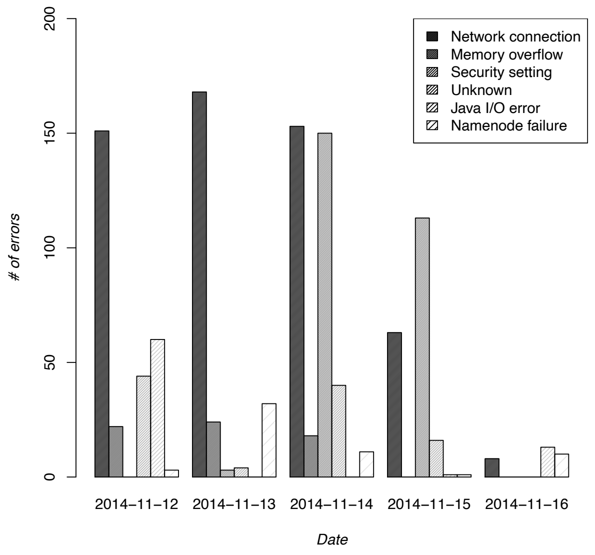


Figure 5: Different types of error in Hadoop cluster during 5 days interval.

Errors such as operational, software, configuration, resource and hardware are present in Hadoop cluster. In this analysis, hardware failure was not considered. Operational, software and resource errors are taken into consideration to detect a failure in the software level of Hadoop cluster. Operation errors include missing operations and incorrect operations. Operation errors are easily identified in log messages by operations: $HDFS\_READ$, $HDFS\_WRITE$, etc. Resource errors (memory overflow, node failure) refer to resource unavailability occurring in the computing environment. Software errors (Java I/O errors, unexceptional) refer to software bugs and incompatibility. These types of error are detected on different DataNodes. Log messages are classified into six different types of error: Network connection, Memory overflow, Security setting, Unknown, Java I/O error, NameNode failure as shown in Figure 5. Errors like network connection and security setting are most occurring errors in the Hadoop cluster. In the pre-processing step, each log message is tagged with certain error ID and using the clustering algorithm like k-means, different types of error are analyzed.

4.2 Predicting failure state in Hadoop cluster.

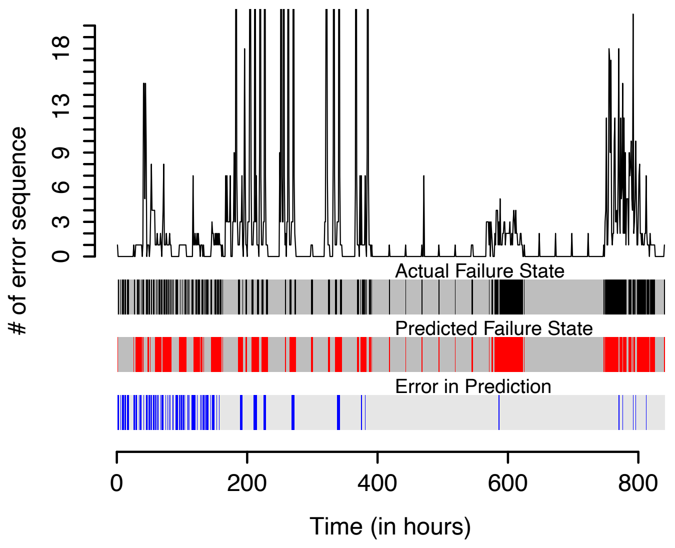


Figure 6: Using HMM to predict failure and normal state. Error sequences are the observation or observable state of HMM and gray-black bar indicate the hidden state, gray line indicate non-failure state and black indicate failure state. Similarly, red line indicates predicted failure state. And blue line indicates prediction error.

Different types of error are used as input observation in our model i.e.$O = \{O_1, O_2....O_n\}$. O_1 to O_6 are error sequences, and O7 is a non-error sequence from the log template. This observation is used in the model $\lambda = (\pi, A, B)$ to detect $S = \{Healthy, Failure\}$. Based on the error sequence in the HMM model, with the help of Viterbi algorithm, hidden state sequences are generated which are shown in gray and red line in figure 6. The red line indicates failure state and gray indicates non-failure state. Similarly, black and gray line shows actual failure state. Based on the probability of the previous state and HMM parameters, the failure, and non-failure states are determined. Error sequences are predicted using EM algorithm, and based on predicted error sequences, hidden states (failure or non-failure) are predicted. Error in prediction is calculated by differencing the actual and predicted value as shown in the graph. At first step, our model is trained from the previous record. As the time passes, the model gets more accurate. The training of the model depends solely on the initial parameter. For this example, initial parameters are calculated from the past record. That is why this model has similar behavior from the initial point, but not an accurate prediction.

The models' ability to predict failure precisely is evaluated by four metrics: precision, recall, false positive rate and F-measure. These metrics are frequently used for prediction evaluation and have been used in many relevant researches as in e.g. [25]. Four metrics are defined in table 1:

Metric	Definition	Calculation
Precision	$p = \frac{TP}{TP+FP}$	0.93
Recall	$r = \frac{TP}{TP+FN}$	0.91
False positive rate	$fpr = \frac{FP}{FP+TN}$	0.091
F-measure	$F = \frac{2pr}{p+r}$	0.92
Accuracy	$accuracy = \frac{TP+TN}{TP+FP+FN+TN}$	0.91

Table 1: Definition of metrics.

From the above observation, we used log entries of 800 hours out of which; first 650 hours is used for training and last 150 hours is used for prediction. In total, we have 24000 observations for 150 hours of prediction time. Different cases for prediction is shown in table 2. The accuracy of the model is 91.25 $\%(9000 + 12900/24000)$. And the precision and recall is 0.93 and 0.91 respectively.

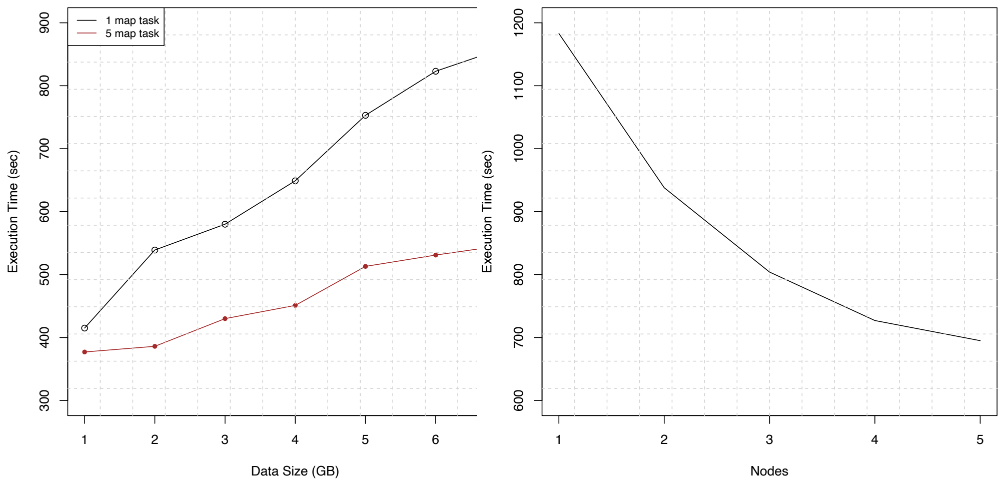
Predicted failure state	Predicted non-failure state
TN: 9000	FP: 900
FN: 1200	TP: 12900

Table 2: Observation for different cases.

Higher precision ensure fewer false positive errors, while a model with high recall ensures lesser false negative errors. Ideal failure prediction model would achieve higher precision and recall value, i.e. precision = recall = 1. However, both high recall and precision are difficult to obtain at the same time. They are often inversely proportional to each other. Improving recall in most cases lowers the precision and vice-versa. F-measure ensures that the model is accurate or not. It provides both precision and recall are reasonably high. In HMM method, a threshold value allows the control of true positive rate and false positive rate. This is one of the big advantages of HMM, method over other techniques.

4.3 Scalability

To test the implementation of MapReduce HMM model in the cluster, we fixed the number of nodes in the cluster to be 6. And, then tested HMM by varying the number of data size from 1 GB (85 million error sequences) to 7 GB (571



(a) Scalability with increases in data size (b) Scalability with increases in cluster size.

Figure 7: Scalability of failure prediction algorithm

million error sequences). Figure 7a demonstrates the scalability of the algorithm. It shows a steady increase in execution time with the growth in data size. The brown and black lines in the graph represent parallel and sequential execution of map task. It is obvious that parallel execution outperform.

In the figure 7b, we set the number of nodes participating in the MapReduce calculations to 1, 2, 3, 4, or 5. The algorithm was then tested on the dataset of size 5 GB (138 million error sequences). The experimental result shows that the execution time improves with an increase in the number of nodes. This increase can significantly improve the system processing capacity for the same scale of data. By adding more nodes to the system, the performance improves and computation can be distributed across the nodes [20]. We conclude that this performance improvement would be even more noticeable with large-scale data involving many more nodes. The ideal scalability behavior would illustrate a linear line in the graph. However, it is impossible to realize this ideal behavior due to many factors such as network overheads.

5 Conclusion

As failures in cluster systems are more prevalent, the ability to predict failures is becoming a critical need. To address this need, we collected Hadoop logs from Hadoop cluster and developed our algorithm on the log messages. The messages in the logs contain error and non-error information. The messages in the log were represented using error IDs, which indicate message criticality. This paper introduced a novel failure prediction method using distributed HMM method over distributed computation. The idea behind this model is to identify the error pattern that indicates an upcoming failure. A machine learning approach like HMM has been proposed here, where the model is trained first using previously pre-processed log files and then it is used to predict the failures. Every log entry is split into equal intervals, defined by sliding window. These entries are separated into error sequence and non-error sequence. Training of the model is done using past observation. Viterbi's algorithm does the prediction of hidden state. Experimental results using Hadoop log files provide an accuracy of 91% and F-measure of 92% for 2 days of prediction time. These results indicate that it is promising to use the HMM method along with MapReduce to predict failure.

References

1. Apache: Apache flume (2010), `https://flume.apache.org/FlumeUserGuide.html`
2. Baum, L.E., Eagon, J., et al.: An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. Bull. Amer. Math. Soc 73(3), 360–363 (1967)
3. Box, G.E., Jenkins, G.M., Reinsel, G.C.: Time series analysis: forecasting and control. John Wiley & Sons (2013)
4. Chang, H., Kodialam, M., Kompella, R.R., Lakshman, T., Lee, M., Mukherjee, S.: Scheduling in mapreduce-like systems for fast completion time. In: INFOCOM, 2011 Proceedings IEEE. pp. 3074–3082. IEEE (2011)
5. Daidone, A., Di Giandomenico, F., Bondavalli, A., Chiaradonna, S.: Hidden markov models as a support for diagnosis: Formalization of the problem and synthesis of the solution. In: Reliable Distributed Systems, 2006. SRDS'06. 25th IEEE Symposium on. pp. 245–256. IEEE (2006)
6. David: anarchyape (2013), `https://github.com/david78k/anarchyape`
7. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6. pp. 10–10. OSDI'04, USENIX Association, Berkeley, CA, USA (2004), `http://dl.acm.org/citation.cfm?id=1251254.1251264`
8. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological) pp. 1–38 (1977)
9. Durbin, R.: Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge university press (1998)
10. Faghri, F., Bazarbayev, S., Overholt, M., Farivar, R., Campbell, R.H., Sanders, W.H.: Failure scenario as a service (fsaas) for hadoop clusters. In: Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management. p. 5. ACM (2012)
11. Fahad, A., Alshatri, N., Tari, Z., ALAmri, A., Y Zomaya, A., Khalil, I., Foufou, S., Bouras, A.: A survey of clustering algorithms for big data: Taxonomy & empirical analysis (2014)
12. Fonseca, R.: X-trace (2010), `https://github.com/rfonseca/X-Trace`
13. Fulp, E.W., Fink, G.A., Haack, J.N.: Predicting computer system failures using support vector machines. In: Proceedings of the First USENIX Conference on Analysis of System Logs. pp. 5–5. WASL'08, USENIX Association, Berkeley, CA, USA (2008), `http://dl.acm.org/citation.cfm?id=1855886.1855891`

14. Hassan, M.R., Nath, B., Kirley, M.: A fusion model of hmm, ann and ga for stock market forecasting. Expert Systems with Applications 33(1), 171–180 (2007)
15. Huang, X., Acero, A., Hon, H.W., Foreword By-Reddy, R.: Spoken language processing: A guide to theory, algorithm, and system development. Prentice Hall PTR (2001)
16. Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R., Wu, A.Y.: An efficient k-means clustering algorithm: Analysis and implementation. IEEE Trans. Pattern Anal. Mach. Intell. 24(7), 881–892 (Jul 2002), http://dx.doi.org/10.1109/TPAMI.2002.1017616
17. Konwinski, A., Zaharia, M., Katz, R., Stoica, I.: X-tracing hadoop (2008)
18. Liang, Y., Zhang, Y., Sivasubramaniam, A., Sahoo, R.K., Moreira, J., Gupta, M.: Filtering failure logs for a bluegene/l prototype. In: Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on. pp. 476–485. IEEE (2005)
19. Marcos, P.d.B.: Maresia: an approach to deal with the single points of failure of the mapreduce model (2013)
20. Mccreadie, R., Macdonald, C., Ounis, I.: Mapreduce indexing strategies: Studying scalability and efficiency. Inf. Process. Manage. 48(5), 873–888 (Sep 2012), http://dx.doi.org/10.1016/j.ipm.2010.12.003
21. Ng, F.D.T.E.: Analysis of hadoopâĂŹs performance under failures. Rice University
22. Plötz, T., Fink, G.A.: Markov Models for handwriting recognition. Springer (2011)
23. Rabiner, L.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2), 257–286 (1989)
24. Sahoo, R.K., Sivasubramaniam, A., Squillante, M.S., Zhang, Y.: Failure data analysis of a large-scale heterogeneous server environment. 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) 0, 772 (2004)
25. Salfner, F., Malek, M.: Using hidden semi-markov models for effective online failure prediction. In: Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on. pp. 161–174. IEEE (2007)
26. SWIMProjectUCB: Swimprojectucb/swim (2012), https://github.com/SWIMProjectUCB/SWIM
27. Tai, A.H., Ching, W.K., Chan, L.Y.: Detection of machine failure: Hidden markov model approach. Computers & Industrial Engineering 57(2), 608–619 (2009)
28. Tan, J., Pan, X., Kavulya, S., Gandhi, R., Narasimhan, P.: Salsa: Analyzing logs as state machines. WASL 8, 6–6 (2008)
29. Teoh, T.T., Cho, S.Y., Nguwi, Y.Y.: Hidden markov model for hard-drive failure detection. In: Computer Science & Education (ICCSE), 2012 7th International Conference on. pp. 3–8. IEEE (2012)
30. Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. Information Theory, IEEE Transactions on 13(2), 260–269 (1967)
31. Wang, F., Qiu, J., Yang, J., Dong, B., Li, X., Li, Y.: Hadoop high availability through metadata replication. In: Proceedings of the first international workshop on Cloud data management. pp. 37–44. ACM (2009)
32. White, T.: Hadoop: The Definitive Guide. O'Reilly Media, Inc. (2012)
33. Wilson, A.D., Bobick, A.F.: Parametric hidden markov models for gesture recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions on 21(9), 884–900 (1999)
34. Zawawy, H., Kontogiannis, K., Mylopoulos, J.: Log filtering and interpretation for root cause analysis. In: ICSM. pp. 1–5. IEEE Computer Society (2010), http://dblp.uni-trier.de/db/conf/icsm/icsm2010.html#ZawawyKM10