Failure Prediction And Detection In Cloud Datacenters

Purvil Bambharolia, Prajeet Bhavsar, Vivek Prasad

Abstract: Cloud computing is a novel technology in the field of distributed computing. Usage of Cloud computing is increasing rapidly day by day. In order to serve the customers and businesses satisfactorily, fault occurring in datacenters and servers must be detected and predicted efficiently in order to launch mechanisms to tolerate the failures occurred. Failure in one of the hosted datacenters may propagate to other datacenters and make the situation worse. In order to prevent such situations, one can predict a failure proliferating throughout the cloud computing system and launch mechanisms to deal with it proactively. One of the ways to predict failures is to train a machine to predict failure on the basis of messages or logs passed between various components of the cloud. In the training session, the machine can identify certain message patterns relating to failure of data centers. Later on, the machine can be used to check whether a certain group of message logs follow such patterns or not. Moreover, each cloud server can be defined by a state which indicates whether the cloud is running properly or is facing some failure. Parameters such as CPU usage, memory usage etc. can be maintained for each of the servers. Using this parameters, we can add a layer of detection where in we develop a decision tree based on these parameters which can classify whether the passed in parameters to the decision tree indicate failure state or proper state.

Index Terms: Cloud computing; failure prediction; failure detection; cloud datacenters; probability and statistics; Bayesian probability; machine learning

1 INTRODUCTION

Cloud computing is one of the new buzzwords in the world of computer science and engineering. One of the main features that cloud service providers offers its customers is the ability to pay per use. Cloud computing also incorporates self-service and thus aiming at Autonomic computing. It also aims at providing businesses with a scalable and adaptive environment. Businesses are provided with various deployment models to choose from. Some of the deployment models include public cloud. Public cloud hosts all the data and resources on public servers which can be accessed by everyone. Private cloud is another type of model where the data is hosted on datacenters located on premise locations. Community cloud is another type of model where a set of organizations or businesses share a cloud datacenter. Hybrid model is a solution to insufficient capacity of the private cloud. Organizations are provided with an ability to subscribe to public cloud services when facing insufficient resources. This is termed as “cloud bursting” [1]. According to [1], cloud provides myriad of advantages which the local data centers cannot. These includes scalability, data security, fault tolerance, pay per use, dynamic resource allocation and much more. As a result, many giant organizations and local businesses are opting to migrate the local data in datacenters onto the cloud. It provides a level of abstraction and thus obviates the need for the users to manage the complex hardware and infrastructure. Being said that, there are many Quality of Service (QoS) parameters which the cloud service providers (CSPs) need to provide as per the Service Level Agreements (SLAs) with their customers. One important parameter is availability of resources [2].

2 LITERATURE SURVEY

Author Name	Paper Name	Year Published	Problems	Open Research Issues
Yukihiro Wantabe, Hiroshi Otsuka, Masataka Sonoda, Shinji Kikuchi and Yasuhide Matsumoto	Online Failure Prediction in Cloud Datacenters by Real-Time Message Pattern Learning	2012	Various types of message logs communicated between components	In the future, accuracy can be improved by including more parameters.
Qjang Guan, Ziming Zhang and Song Fu	Ensemble of Bayesian Predictors and Decision Trees for Proactive Failure Management in Cloud Computing Systems	2012	Assumption that training dataset is labeled;	In future, integrating two failure management approaches will lead to better dependability.
Husanbir Pannu, Jianguo Liu, Qiang Guan and Song Fu	AFD: Adaptive Failure Detection System for Cloud Computing Infrastructures	2012	The runtime performance data is usually unlabeled and thus a prior failure history is not always available.	Integrating the proactive and reactive failure management approaches like check pointing and redundant execution can increase the fault detection accuracy.

Yukihiro Watanabe, Hiroshi Otsuka and Yasuhide Matsumoto	Failure Prediction for cloud datacenter by Hybrid Message Pattern Learning	2014	When applying failure prediction to the cloud datacenters, incorrect detection can result in execution of unnecessary tasks and additional costs.	Various kinds of data from the systems such as messages, performance data and configurations can be used to achieve more precise failure prediction.
Dinh-Mao Bui, Thien Huynh and Sungyoung Lee	Fuzzy Fault Detection in IaaS Cloud Computing	2016	Most of the existing prediction methods are supervised learning models. However, the labeled dataset is not always available in real world cloud computing systems.	Probabilistic approach can be used to increase the ability of self-decision making for fault tolerance system.

Availability directly depends upon how fast the cloud infrastructure can detect any faults and take necessary steps to troubleshoot the problem. It is a major challenge for service providers to provide stable service or else it may cause huge monetary loss for organizations. The time required to repair the fault is also a critical parameter in failure management. However, failure detection is often a difficult task due to complexity and dynamic nature of the cloud. Hence as compared to non-virtualized systems and local datacenters, the troubleshooting procedure is more time-consuming and complex. Therefore, an effective approach is to detect a failure in its nascent stages. Hence preventive measures can be undertaken by the system administrator to avoid a disastrous failure before it occurs. This technique is called failure prediction. It takes use of runtime statistical information and the information of previously observed failures to forecast future failure occurrences in the cloud. Large amount of monitoring data is collected to track the status of cloud during its operation. Such data includes the software log files, traffic in the network, system audit events, virtualization information, etc. This monitoring data is so large that it is impossible for us to manually deduce the cloud status from it. Besides due to dynamicity of cloud, it is common that the state information and the networks' load might change. Thus overall fault detection and prediction in cloud is a challenging task and it is vital that the service providers take necessary steps to effectively troubleshoot it in order to provide stable service to their users. Most of the conventional failure detection and prediction models are based on statistical learning techniques. These models are supervised learning models and they take into consideration various performance attributes to approximate the dependency of failure occurrences. The assumption on which these models are based is that the training dataset is labeled, i.e. for each data point used to train the models, the designer knows well in advance if that data point corresponds to a failure or normal execution. However for newly deployed systems on cloud, the labeled data set is not always available. In supervised learning models, the undetected failure occurrences are never taken into consideration for failure prediction. Thus it is a challenging task on how to accurately predict failures given the dynamic nature of cloud. In this paper, we propose a method which consists of two modules. The first module would be used to predict the failure using message logs passed between various components of cloud as proposed in [1]. The secondary module will be used to detect failure at an estimated time. The prediction model would predict that failure is going to occur and certain processes would be launched when a failure is predicted, so as to reduce the impact of the failure. But what if the prediction is incorrect. In such cases, we can always verify that our prediction is correct using failure detection. We let the system launch impact reducing processes. We then detect whether a failure has actually occurred or not using the state information of the cloud. If failure is detected, we let the impact reducing mechanisms function normally, but if failure is not detected, we roll back the processes to previous state and shutdown the processes launched to reduce failure impact.

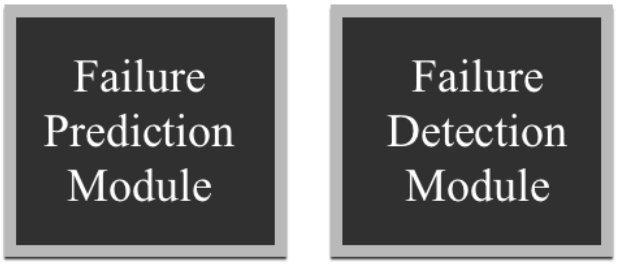

Fig 1. *Two modules part of the model presented*

2 Problem Statement

As discussed earlier, the large scale and dynamic nature of cloud has added extra complexity when it comes to fault detection and management. Also, how to overcome it is also a grave concern for the CSPs. While it is true that effective fault detection and prediction is critical, one should also understand the reasons that led to the fault. Some of the reasons for the same are human error, cloud provider downtime, extreme spikes in customer demand, third party service failures, storage failures and much more. Thus, the service providers should take effective steps to mitigate these failure reasons. These faults may lead to unavailability of resources and even downtime in web applications and web services which in turn can affect large population of customers and even severe economic loss. A very famous example to support this claim is of Amazon which was down for almost 45 minutes due to an unexpected fault and eventually led to about 5-million-dollar loss in transactions. Due to the dynamic nature and changing workload, it is extremely difficult from the data collected to predict fault. Also the runtime performance data is usually unlabeled and thus prior failure history is not always available. Another challenge is that the cloud hardware and the software components are constantly replaced or updated. Hence it is necessary for the fault detection mechanism in use to distinguish accurately the normal cloud variation and real time failures. Also, if the cloud service provider can predict the

failure, it would act as a cherry on the icing on the cake. Also, to increase consumer and customer satisfaction which in turn will increase revenue and sales for the CSP, a CSP must proactively handle and mitigate the effects of failure across the cloud system. Failure in a large scale cloud system may also lead to huge economic losses. These losses can be prevented by handling the failures faced by the cloud systems in real time.

3 SOLUTION STATEMENT

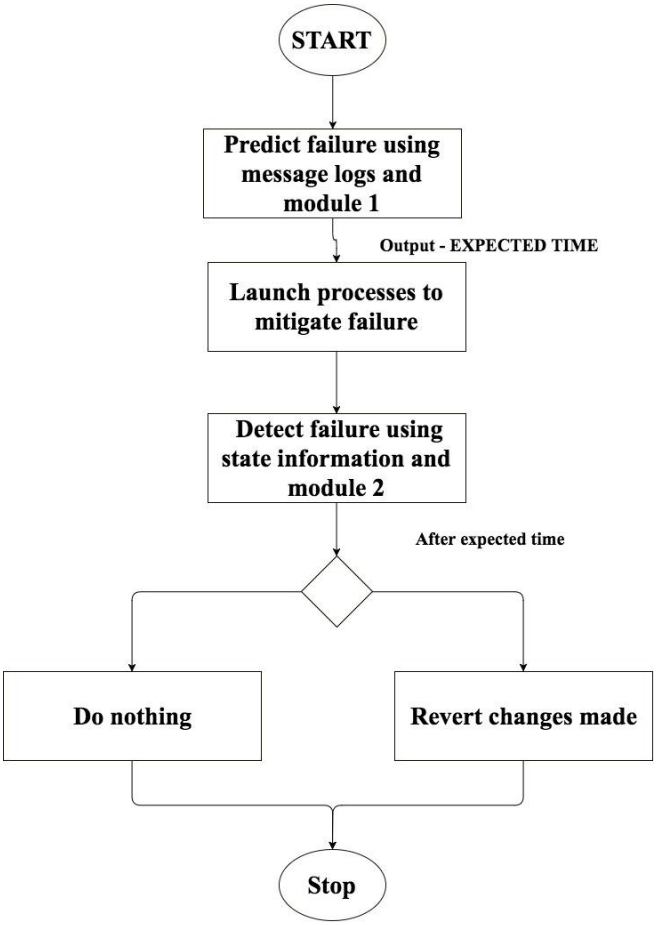

Fig 2. *Flowchart for the model presented*

The overall model of detecting and predicting failures proposed in this paper includes four major steps. First step includes predicting a failure in the cloud system with expected time using the model proposed in the paper. Step two includes launching certain processes that can be used to mitigate the effects of failure. It also includes launching mechanisms such as backup processes, saving cloud system state, etc. that can reduce the extent to which the failure can spread in a cloud system. Step three includes detecting failure using the state information after the expected time has elapsed. Step four includes an IF-ELSE condition. If a failure is detected after the elapsed expected time as predicted by our predictor model, if failure is not detected we revert back the processes we launched when the failure was predicted.

A. Fault Prediction

As discussed above, the module 1 of the proposed model will predict the failure in the cloud systems. The module will predict the failure by fetching and analyzing the pattern of messages of real time. In order to forecast failure effectively, it is critical to classify the messages accurately irrespective of the format. Also we need to keep track of the configuration settings as they change dynamically and the predicted result should consider the order of the messages collected in logs. The architecture for the real time message pattern learning which meets the above requirements is depicted below:

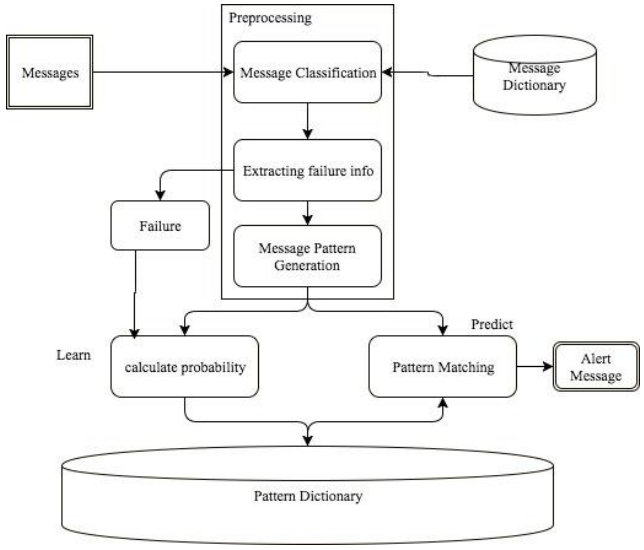

Fig 3. *Failure Prediction Architecture*

As shown above, the architecture consists of three phases: (a) preprocess phase which classifies input messages and derives patterns from it, (b) learning phase calculates the probability of message pattern coexisting with failure and (c) prediction phase, which compares the real time messages with the learned messages to forecast the possibility of failure.

a) Preprocess

Firstly, we derive the message patterns from the recorded messages in a message logs. This phase involves three steps: message classification, extracting the failure information and message pattern generation.

1) Message Classification

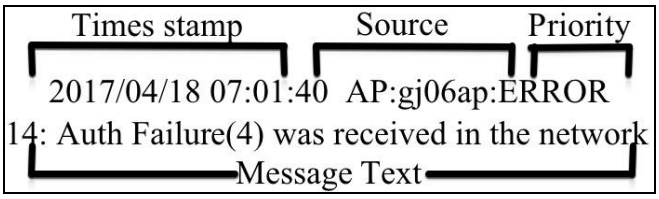

Fig 4. *Message text example*

As shown in the figure above, a given message is divided into four fields: timestamp, message source, priority and message text. In this step, the messages are classified based on their similarity index. Thus, two messages will be classified into single group if they share many common words. Message dictionary is used to classify message texts. An example of message dictionary is shown in table below which contains two columns – “message type”, which is a unique identification

number for a message type and "words", it contains a set of words corresponding to a particular message type.

Message Type	Words
1	Stand-by, interface failure, recovered, vlan, etc.
2	Send-mail, space insufficient, need for space, etc.
3	Auth. Failure, TRAP, generic, timestamp.
. .	…

Table 1. *An Example of Message Dictionary*

Consider a set of words Di corresponding to the i-th message type. Let d_{i1}, d_{i2}, … be the words which are classified to the i-th message type. Now given a message e_j which needs to be classified. Firstly, e_j is split into set of words W_j = { w_{j1}, w_{j2}, …, w_{jn} } where n is the number of words in the message text. Then we find S_{ij} which represents the degree of similarity between Di and W_i.

$$S_{ij} = \sum_{k=1}^{n} s_{ik} \quad where \quad \begin{pmatrix} s_{ik} = 1 & if & w_{jk} \in D_i \\ s_{ik} = 0 & if & w_{jk} \notin D_i \end{pmatrix}$$

After calculating the similarity scores for all i, we choose the candidate for classification as set of words Dc having maximum S_{cj}. After this we will be having the candidate message type c with score S_{cj}. From this, similarity S can be calculated as:

$$S = \frac{S_{cj}}{n}$$

A threshold value $th_{classify}$ is used to decide whether the given message should be classified into an existing one or a newly created one. Hence if S > $th_{classify}$, the given message e_j is classified as class type c and if it has some new words not in D_c, then they are added. Or else we create a new entry in the message dictionary.

2) Extracting the failure information

As discussed above, messages generally include a field which signifies priority information and helps the administrators to handle the messages according to their severity. The messages with severity – Error, Critical, Alert or Emergency are considered as the events showing failure occurrence. Information for the failure is extracted once the message is classified as a failure. As shown in table II, the failure information table consists of two parameters, timestamp and description of failure and this information is used further in learning process.

ID	Timestamp	Type of failure
75	2017/04/18 10:40:55	Code 10 – Fault in availability of process
76	2017/04/18 10:41:30	Code 20 – Memory insufficient

Table 2. *Examples of failure information*

3) Message pattern generation

A message pattern is defined as a set of message types in the message window. The message pattern can be expressed as a sequence of messages by either considering or ignoring their order. Comparing the efficiency of performance methods with binary vector (it ignores the count of each type of message in the window) and event vector (considers the count). From their experiments, it was found that keeping track of count of each message type has no advantage. Therefore, the message pattern is expressed by using a binary set of message types in the window instead of using event vector. Table III shows the message pattern record where the k-th entry is expressed as mp_k, a message pattern and timestamp are expressed as $m(mp_k)$ and $t(mp_k)$ respectively.

b) Learning Message Patterns

By observing the message pattern, we can estimate the probability that a failure occurs in a certain period using Bayes' theorem:

$$P(G_i \mid mp_k) = \frac{P(G_i)P(m(mp_k) \mid G_i)}{P(G_i)P(m(mp_k) \mid G_i) + P(\neg G_i)P(m(mp_k) \mid \neg G_i)}$$

Where
P(X): Probability of occurring event X
P(X|Y): Conditional probability of X given Y has occurred
G_i: Set of event corresponding to i-th type of failure
$P(G_i|mp_k)$: Probability that a failure F_i occurs in a predictive period after timestamp of mp_k The above equation can be simplified as given below by considering the frequency of occurrence of each event type:

$$P(G_i \mid mp_k) = \frac{n(m(mp_k) \cap G_i)}{n(m(mp_k))}$$

$$n(m(mp_k) \cap G_i) = \sum_{l=1}^{k} h(mp_l, mp_k, F_i)$$

Where

$$h(mp_l, mp_k, F_i) = \begin{cases} 1 & if\ m(mp_k) = m(mp_l) \quad and \\ & (t(F_i) - d) \le t(mp_l) < t(F_i) \\ 0 & otherwise \end{cases}$$

$$h(mp_l, mp_k, F_i) = \begin{cases} 1 & if\ m(mp_k) = m(mp_l) \quad and \\ & (t(F_i) - d) \le t(mp_l) < t(F_i) \\ 0 & otherwise \end{cases}$$

$t(F_i)$: timestamp of F_i
d: predictive period length

$$n(m(mp_k)) = \sum_{l=1}^{k} g(mp_l, mp_k)$$

Here

$$g(mp_l, mp_k) = \begin{cases} 1 & if\ m(mp_k) = m(mp_l) \\ 0 & otherwise \end{cases}$$

Based on the above equations, we can estimate the co-occurrence probability of an observed message pattern and a failure occurrence using the message pattern dictionary. An

example of the pattern dictionary is shown in the table below. Here numerator represents the number of message pattern observed in predictive period of failure F_i and denominator is the total number of message pattern. When mp_k is added to the message pattern record, $P(G_i|mp_k)$ for all F_i is updated and the number of patterns $n(m(mp_k))$ is increased. Firstly, the predicted period is calculated for the input message type followed by $m(mp_k) \cap G_i$ which are timestamped message patterns. As a result, the number of pattern in this predictive period is also increased. Finally, in the pattern dictionary, posterior probability $P(G|mp_k)$ is updated. The learned result can be used by the predicting phase.

c) Predicting Failure

Failures can be predicted using the message patterns and probability from the pattern dictionary. Given a message pattern $m(mp_k)$, the pattern dictionary is referred and the column matching the message pattern is selected. Then, probability for each type of message failure is calculated. If the probability for any message type of failure is higher than a threshold, we conclude them as a possible sign of failure and necessary steps are undertaken to resolve them.

B. Fault Detection

Here we discuss about the module 2 of the model we propose for failure detection and prediction. The idea is to develop a probabilistic model on the basis of some of the runtime state parameters such as temperature of the cloud system, CPU usage of the system, memory usage of the system, CPU idle time, number of I/O operations per unit time, etc. There can be two types of situations. One, in which the cloud system is behaving normally, and the other being a state in which some failure has occurred. Hence, we can have 2 classes, class 1 for normal behavior and class 2 for abnormal behavior [4]. The problem with making such classes is that there may not be enough data relating to class 2 [4]. Saving the state information of the cloud system will be a part of a module in the cloud system. If this module itself fails or progressively fails due to the failure of some other module, the state information won't be saved for further analysis through which we can calculate probabilities related to class 2 for each of the runtime state parameters. Hence, there may not be enough data related to class 2 for the supervised learning algorithm to estimate probabilities related to class 2 [5]. To solve this problem, we can use a semi-supervised approach. We can use the unbalanced dataset of the state information to calculate probabilities relating to class 1 which is a majority class [4]. We can then classify current state information as normal or abnormal based on these probabilities related to majority class. A data point would relate to class 2 if its probability for belonging to class 1 would be too low. A semi-supervised approach would include building a model based on the majority class. Data related to state information are collected from the cloud system and transformation techniques are applied to it, so as to, maintain them in a uniform format. Features of the dataset would be the state parameters related to cloud system. For large cloud systems, this dataset may contain thousands of features. Thus, dimensionality reduction is very important factor for failure detection. We apply dimensionality reduction techniques which produces a new dataset with only the most relevant set of features [4]. In this paper, we discuss one of the techniques that can be used to reduce the number of features in the dataset. The technique is termed as 'Relevance Deduction'. Let the dataset be represented by features X_1, X_2, X_3, …., X_n. The intuition is to calculate similarity between each pair of features. Consider two features X_a and X_h. To calculate mutual information which represents covariance between the two features, we use the following formula [5].

$$MutualInfo(X_a, X_b) = \sum_{x_a} \sum_{x_b} p(x_a, x_b) log(\frac{p(x_a, x_b)}{p(x_a)p(x_b)})$$

If mutual information between two features turns out to be a minimum, we can conclude that the two features are completely independent. In the opposite case where, mutual information between two features turns out to be maximum, we can conclude that the pair of features are indeed same. The objective is to select a subset of original features on the basis of mutual information. To exclude unnecessary features, we can calculate index values for each feature. Features with high index values can be neglected [5].

$$Index(X_a) = \sum_{b=1}^{i=n} (MutualInfo(X_a, X_b))$$

Now for the purpose of detecting a failure, we design and develop a statistical model *m* with reduced dimensionality which takes a data point as input and outputs its probability. Parameters related to the model are learned in an unsupervised manner. A data point d is detected as a failure only if output of the model *m(d) < threshold.* Threshold value can be determined based on assumptions and past experience. An input data point d is classified as normal behavior or abnormal behavior based on its probability of belonging to normal behavior class which is the majority class. We develop a probabilistic model consisting of a group of Bayesian sub models. Each sub model is a non-parametric data model [4]. Frequency counts of the training data are used to calculate probability distribution. Each sub model is linked with an estimated probability *p(i),* where i is index associated with the sub model. The probability for a data point d can be calculated by using the following formula, where n is the number of sub models in the model [4].

$$p(d) = \sum_{i=1}^{i=n} p(d/i)p(i)$$

Due to the above formula, all the sub models are allowed to contribute to the probability of the input data point [4]. This approach allows the model to fit the historical data much more accurately. After the dimensionality of the training data set is reduced, the features in each sub model would be conditionally independent of each other [4]. Following formula is used for calculating *p(d/i)* for a data point d with l number of features after dimensionality reduction as discussed earlier.

$$p(\frac{d}{i}) = \prod_{j=1}^{j=l} p(\frac{d_j}{i})$$

We use Bayesian's expectation maximization algorithm, also known as EM algorithm, to calculate the probability that a feature d_j takes a given value v based on frequency count in the training set.

$$p(d_j = v|i) = count(d_j = v \text{ and } i)/count(i)$$

$$p(i) = \frac{count(i)}{\sum_{n \epsilon submodels} count(n)}$$

In the above equations, count( ) is the number of tuples satisfying a particular condition in the given data set. Following formulas can be used for calculating subparts of the above equations [5].

$$count(d_j = v \text{ and } i) = \sum_{d \in training set}^{p} (i/d) I(d_j, v)$$

$$count(i) = \sum_{d \in training set} p(i/d)$$

Values of $I(d_j, v)$ is equal to 1 if $d_j = v$ else $I(d_j, v)$ is equal to 0. To train this model which is in turn a group of sum models, we use expectation maximization algorithm [5]. We do not discuss about the algorithm in this paper.

4. Conclusion

We conclude this paper by attempting to solve the complex and complicated problem of proactive management of failures in a large scale cloud system. Proactive management of failures must be at the highest priority in terms of problems to be solved in the hands of the CSPs. Failure prediction and detection can not only increase consumer satisfaction but, it can also increase revenue sales and decrease economic loses.

5. Future Scope

This research can be further continued by focusing on improving the accuracy of the prediction model. Other advance prediction models such as neural networks, recurrent neural networks, etc. may be adopted to increase the accuracy of the prediction model. Focusing on increasing accuracy of prediction model can lead to better advancement in proactive failure management. This research paper can further be extended by researching around the topic of calculating the expected down time using prediction analysis.

References

[1]. Buyya, Rajkumar, et al. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility." Future Generation computer systems 25.6 (2009): 599-616.

[2]. Zhang, Qi, Lu Cheng, and Raouf Boutaba. "Cloud computing: state-of-the-art and research challenges." Journal of internet services and applications 1.1 (2010): 7-18.

[3]. Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi and Y. Matsumoto, "Online failure prediction in cloud datacenters by real- time message pattern learning," 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, Taipei, 2012, pp. 504-511. doi: 10.1109/CloudCom.2012.6427566

[4]. Guan, Qiang, Ziming Zhang, and Song Fu. "Ensemble of bayesian predictors and decision trees for proactive failure management in cloud computing systems." Journal of Communications 7.1 (2012): 52-61.

[5]. Guan, Qiang, Ziming Zhang, and Song Fu. "Proactive failure management by integrated unsupervised and semi-supervised learning for dependable cloud systems." Availability, Reliability and Security (ARES), 2011 Sixth International Conference on. IEEE,