R| R |S |T
Researchberg Review of Science and Technology
Vol. 3 No. 1
2020

Research

Research Article

Proactive Fault Tolerance Through Cloud Failure Prediction Using Machine Learning

Vamsikrishna Bandari
University of South Australia
Orcid: 0000-0003-4185-3985

Abstract

One of the crucial aspects of cloud infrastructure is fault tolerance, and its primary responsibility is to address the situations that arise when different architectural parts fail. A sizeable cloud data center must deliver high service dependability and availability while minimizing failure incidence. However, modern large cloud data centers continue to have significant failure rates owing to a variety of factors, including hardware and software faults, which often lead to task and job failures. To reduce unexpected loss, it is critical to forecast task or job failures with high accuracy before they occur. This research examines the performance of four machine learning (ML) algorithms for forecasting failure in a real-time cloud environment to increase system availability using real-time data gathered from the Google Cluster Workload Traces 2019. We applied four distinct supervised machine learning algorithms are logistic regression, KNN, SVM, decision tree, and logistic regression classifiers. Confusion matrices as well as ROC curves were used to assess the reliability and robustness of each algorithm. This study will assist cloud service providers developing a robust fault tolerance design by optimizing device selection, consequently boosting system availability and eliminating unexpected system downtime.

Keywords: Cloud Computing, Fault Prediction, KNN, Machine Learning, SVM

Introduction

As the size of the cloud service platform has grown, the number of apps it hosts has grown with it, and the number of variables that might affect a task's completion time has multiplied in complexity. Many failure-jobs occur as a result of factors such as hardware or software breakdown, node failure, and inadequate scheduling resources, increasing task queuing time and decreasing operational efficiency and wasting cluster resources.

Fault tolerance indicates to the scenario in which a service that is functioning abruptly ceases [1], and then another service is poised to take its spot and complete the task from the point where the first service faltered [2]. In most cases, an application's functionalities, which are contained inside the cloud infrastructure itself, are directly related to its level of performance. The efficiency of a cloud computing service is, above and beyond all other characteristics of cloud computing architecture and resources, the much more significant and critical feature of cloud computing [3]. Performance may be described as the execution and achievement of a given job judged against established recognized criteria of accuracy, comprehensiveness, expense, and time. Distributed computing requires that all of the resources be well though and effective in order to achieve high performance rate.

The cloud service company is primarily responsible for ensuring that consumers are satisfied with the reliability and performance of the cloud computing services they get. The performance of cloud computing might suffer for a number of reasons, but the most important ones are the following [4]: Accessibility of service that denotes available risks for distributed online services computing mostly as a consequence of these features, such as overload of frequent services, programming flaws, or distributed DDoS threats [5], [6]. These obstacles are affecting the pace of data transmission as well as the expense also in cloud, which is a significant hurdle that is transfer of data obstacle and through which certain issues raised and these issues are rising of data, clustering of programs and these bottlenecks [7]. Unpredictability in performance refers to the elements that cause performance hazards, such as poor input/output division and the enormous efficiency of distributed network solutions in the cloud. The scalability of storage is an indicator of the challenges that come with installing cloud computing to acquire solutions that demand incredibly extendable continuous storage. Scaling quickly refers to the challenges of swiftly increasing or decreasing resources in response to changes in demand while maintaining compliance with service-level agreements (SLAs) [8], [9].

The primary function of fault tolerance would be to attempt to suppress the existence of system failures, despite the fact that fault tolerance acknowledges the possibility that faults may happen. There are two stages in the fault tolerance process, and they are as follows: Error detection that offers insights into the current state of operations of processes. Error Recovery is the process of attempting to change the incorrect state of the system into a condition that is free of errors [10].

It is possible for the data centers to be situated in a variety of geographic areas. The performance of cloud services will suffer as a result of the data centers being loaded or overwhelmed when serving the requests of an enormous range of client applications operating on the same place at the same time. It is still difficult to accurately measure the performance of user apps while using a range of resources due to the fact that different user applications need distinct setups.

It is vital to supply an adequate quantity of tools to each virtual machine in order to guarantee quality of service. The most important resources that are made available by the cloud are computer cycles, bandwidth, and storage. If the cloud provider delivers enough resources, quality of service will be automatically ensured. Over-provisioning is one method of achieving this goal; however, this method is indeed not energy efficient and therefore results in economic loss for the provider [11], [12]. There should be a balance between overprovisioning and under-provisioning of resources. Over-provisioning involves allocating a greater number of resources than are required, whereas under-provisioning results in breaches of the Service Level Agreement (SLA) [13]. As a result, resources are distributed in a fair and equitable manner. Therefore, the second method of guaranteeing quality of service is to anticipate a fault in performance by tracking resource use.

The primary function of fault tolerance would be to attempt to hide the existence of system failures, despite the fact that fault tolerance acknowledges the possibility that errors may occur. There are two stages in the fault tolerance process, and they are as follows: Error detection that offers insights into the current state of operations of processes. Error Recovery is the process of attempting to change the incorrect state of the system into a condition that is free of errors.

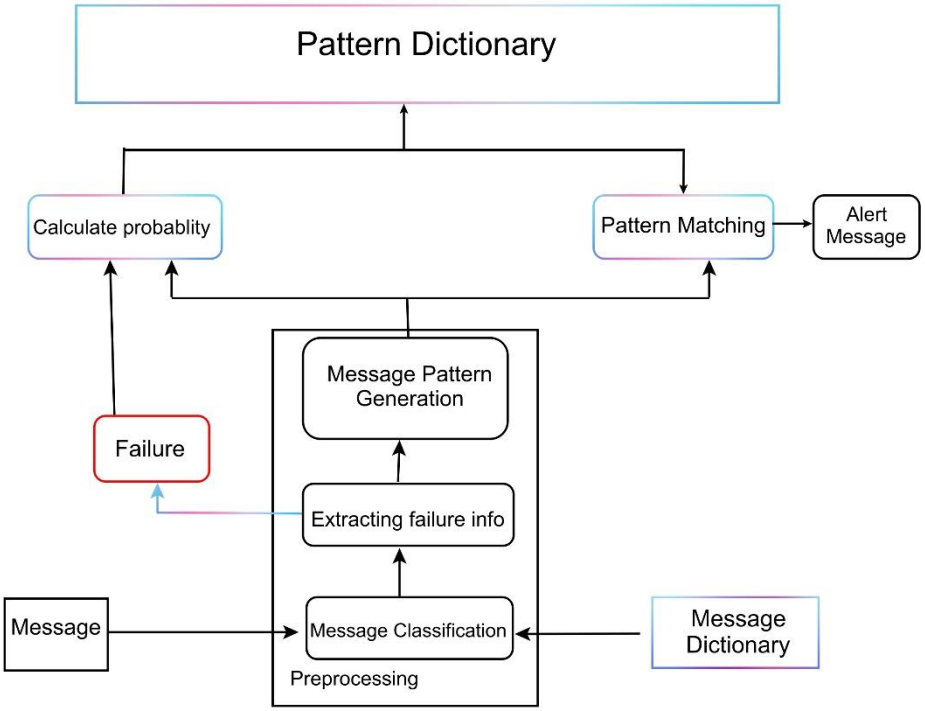

Figure 1. Failure prediction

Fault Tolerance Strategies in Cloud Services

The capacity of a cloud service to adapt to and survive disruptions in service caused by things like hardware failures, software glitches, network congestions, and other unforeseen events is known as fault tolerance. For real-time applications running in the cloud, one may choose between two different fault-tolerance strategies: preventative and responsive.

a. Proactive/Preventative Fault Tolerance:

Avoiding faults and replacing potentially failing parts with known good parts is at the heart of the preemptive fault tolerance idea [1]. Preemptive fault tolerance principles provide the basis for a number of methods, including:

1) Preemptive migrating is a method in which programs are continuously monitored and assessed using a feedback control system.

2) Second, software rejuvenation may be achieved by establishing a regular reboot plan. Each time the system is restarted, it starts in an entirely new state.

b) Reactive/Responsive Fault Tolerance:

Measures taken after failures have already arisen in the Cloud are the focus of the Responsive fault tolerance strategy. Methods such as Checkpointing, Restarting, and Replicating, as well as Task Resubmitting, are all examples of reactive fault tolerance policies [14].

The first technique is called "Checkpointing/Restart," and it saves the current state of a system (which is executing a program) in a global checkpoint [15]. If an error occurs, the system may simply be restored to the last saved state rather than having to restart the program from scratch [16].

The second strategy is called replication, and it is based on one of the most well-known strategies for dealing with failure. The term "replication" refers to the practice of keeping several identical copies of any data or object spread across various servers. By creating several copies of data, replication increases system reliability. Without disrupting the system's operation, tasks may be resubmitted to either the same or a new resource at runtime upon defect detection.

Failure in a cloud system is described as an incident that takes place when the service that is supplied differs significantly from the service that was meant to be given [17]. As the magnitude and sophistication of cloud computing systems continues to expand, there is a pressing necessity for cloud service providers to assure a consistent on-demand tool to their clients even when there is a failure in the system [18]. This will allow them to accomplish the service level agreement that they have with those clients.

Machine learning methods fault prediction

K Nearest Neighbor

The k Nearest Neighbor (k-NN) methodology is a straightforward and extensively used classification method. The strategy is to choose the most similar k number of individuals in k Nearest Neighbors as nearest neighbors to a certain person, then predict the sample's class for a particular model based on the data of the selected neighbors [19]. The k-Nearest Neighbors technique is based on the notion that comparable entities are positioned near to one another. As a result, we select a specific value by taking into account the neighboring locations. We examine the "k" most current observation that are the most comparable to the one whose class we are attempting to anticipate. These "k" statistics from the surrounding region form the basis of our forecast [20].

To eliminate any possible ties, it is customary to use an odd number for the k value. Even values of k might result in the same number of votes. The algorithm's outcomes are influenced by the value of k. As a consequence, the approach varies greatly depending on two factors. The first is how to calculate each data's distance, and the other is how to use or assess the nearest neighbors to predict a certain person's class or category [21].

Let,

$$A=(a_1,a_2,\ldots,a_n) \text{ and } B=(b_1,b_2,\ldots,b_n)\in\mathbb{R}^n$$

There are several techniques for calculating the distance between neighbors. The Minkowski distance is the general case, as follows:

$$D(A,B)=\left(\sum_{i=1}^{n}|a_i-b_i|^p\right)^{1/p}$$

Taking ρ=2, we obtain the Euclidean distance.

$$D_e(A,B)=\left(\sum_{i=1}^{n}(a_i-b_i)^2\right)^{1/2}$$

Logistic regression

For the sake of clarity in the syntax, we will make the assumption that the target accepts values from the data point set (0,1). After the model has been fitted, the predict probability method of Logistic Regression calculates the probability that class,

$$P(y_i = 1 | X_i)$$

will be positive as:

$$\hat{p}(X_i) = expit(X_i m + m_0) = \frac{1}{1 + \exp(-X_i m - m_0)}.$$

In the context of an optimization issue, binary class logistic regression with the regularization term r(m) seeks to find the value of the following cost function that is the lowest possible value [22].

$$\min_w C \sum_{i=1}^{n} \left(-y_i \log(\hat{p}(X_i)) - (1 - y_i) \log(1 - \hat{p}(X_i)) \right) + r(m)$$

Support Vector Machine (SVM)

We wish to identify two parameters $\beta \in \mathbb{R}^p$ and $\alpha \in \mathbb{R}$ such that the following equation is valid for the majority of samples if we have a training set vector $x_i \in \mathbb{R}$ and two classes $y \in \{1, -1\}^n$:

$$\text{sign}(\beta^T \phi(x) + \alpha)$$

In particular, the Support Vector Machine seeks to minimize the following:

$$\min_{\beta, \alpha, d} \frac{1}{2} \beta^T \beta + p \sum_{i=1}^{n} d_i$$

$$\text{subject to} \quad cl_i (w^T \phi(tr_i) + \alpha) \geq 1 - d_i,$$
$$d_i \geq 0, i = 1, ..., n$$

[23]

Where *p* denotes the penalty term and *d* denotes the distance of the samples from the margin border. We get the optimization issue when we express the dual to primal indicated before, and we obtain:

$$\min_c \frac{1}{2} c^T M c - I^T c$$

subject to $y^T c = 0$

$0 \le c_i \le P, i = 1, ..., n$

Where, M is a semidefinite matrix with size n by n.

Decision tree

A decision tree may be used to analyze both continuous and discrete data. It works by splitting the data into groups depending on the attribute deemed most essential within the dataset. The algorithms are in charge of deciding how the decision tree detects this attribute and how the data is divided.

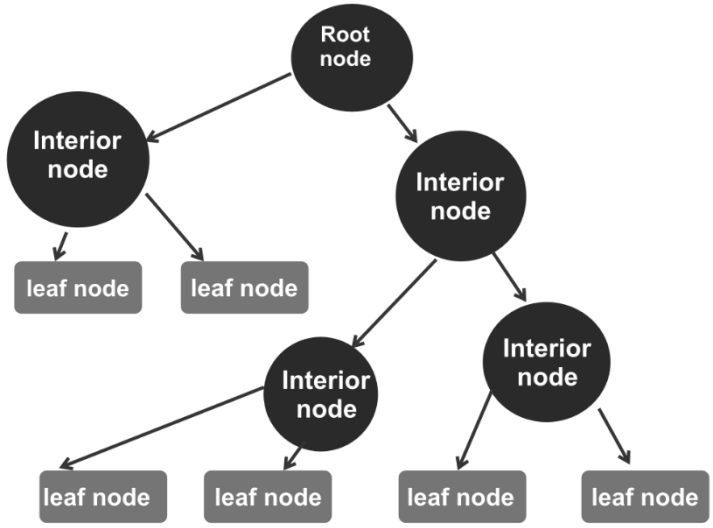

Figure 2: decision tree

The root node is the node that reflects the most significant predictor. Sub-nodes generated as a consequence of splitting are known as decision nodes, whereas nodes that do not split anymore are known as leaf nodes [24].

When employing the decision tree, the dataset is divided into sections that are similar and those that do not intersect. The top portion presents all of the data in a single spot before splitting into two or more segments that continue to separate. This suggests that the process employs a top-down approach. The decision tree processes will continue to run until one of the stop criteria, such as a preset minimum number of observations, is fulfilled [25], [26].

Google Cluster Workload Traces 2019

This research used the data from the workloads operating on eight Google Borg compute clusters in May of 2019. The trace shows how each job that ran in those clusters was submitted, how it was scheduled, and how many resources it used [27].

Each item has a priority, which is expressed here as a tiny number that is mapped into a ranked list of numbers, with 0 being the item with the least priority. Those with greater priorities (those located on higher "tiers") often get priority when it comes to the allocation of resources over items with lower

priorities. There are certain unique priority ranges, as well as applications for them; nevertheless, in general: 2 Free tier (priorities less than 99): these are the priorities with the least amount of weight. When resources are sought at these priority, little or no internal billing is incurred, and the SLOs for getting and maintaining these resources on machines are either very lax or nonexistent. Best-effort Batch (beb) tasks (priority 100–115): jobs that are running at some of these priorities are controlled by the batch scheduler & incur minimal internal charges; they do not have any associated SLOs. Priorities 100–115 are referred to as "batch" jobs. Medium (priorities 116–119): the SLOs for these priorities are situated in the middle ground between those for "free" priorities and production priorities [28]. 3 Production tier (ranges from 120 to 359): These represent the priorities that get the most attention during typical operations. In fact, the Borg cluster scheduler makes an effort to avoid the eviction of latency-sensitive jobs at this priority owing to the excessive allotment of machine resources. Monitoring tier (priorities more than 360): tasks that track the status of other, lower-priority occupations are given these priorities.

Every job and task has a scheduling class, which provides a rough indication of how sensitive the task's execution is to latency. The scheduling class usually would be denoted by a single integer, with the value 3 denoting a job that is more sensitive to latency (for example, responding to user requests that generate revenue), and the value 0 denoting a work that is not involved in production. Take into account that the scheduling class is not a priority, despite the fact that more latency-sensitive tasks typically have high priority: the priority helps determine whether or not something is scheduled on a machine, whereas the scheduling class impacts the machine-local policy for resource availability.

The collection name is hashed before being presented as an encrypted base64-encoded string. This string may then be compared to other names to determine whether or not they are equal. Automated methods may occasionally be used to produce one-of-a-kind job names in order to eliminate potential clashes.

Data from multiple different inner name fields are combined to form the collection logical name, which is an opaque name (e.g.,The vast majority of the numbers in the logical name are changed to a constant string). Logical names are intended to be a kind of partial compensation for such unique names that are produced by automated tools. In this scenario, multiple executions of a single program will typically have the similar logical name. Every record is assigned to a timestamp, that is measured in microseconds from the commencement of the trace period and is stored as a 64-bit number. For example, an event that occurred 20 seconds after the commencement of the trace would be assigned a timestamp equal to 620s.

The CollectionEvents table and the InstanceEvents database, respectively, define the life cycle of collections and instances. Typically, each task that makes up a job will carry out the identical binary execution, complete with the same set of parameters and resource request. Programs that are required to do many sorts of activities that each need a varied amount of resources often carry out their operations as independent jobs.

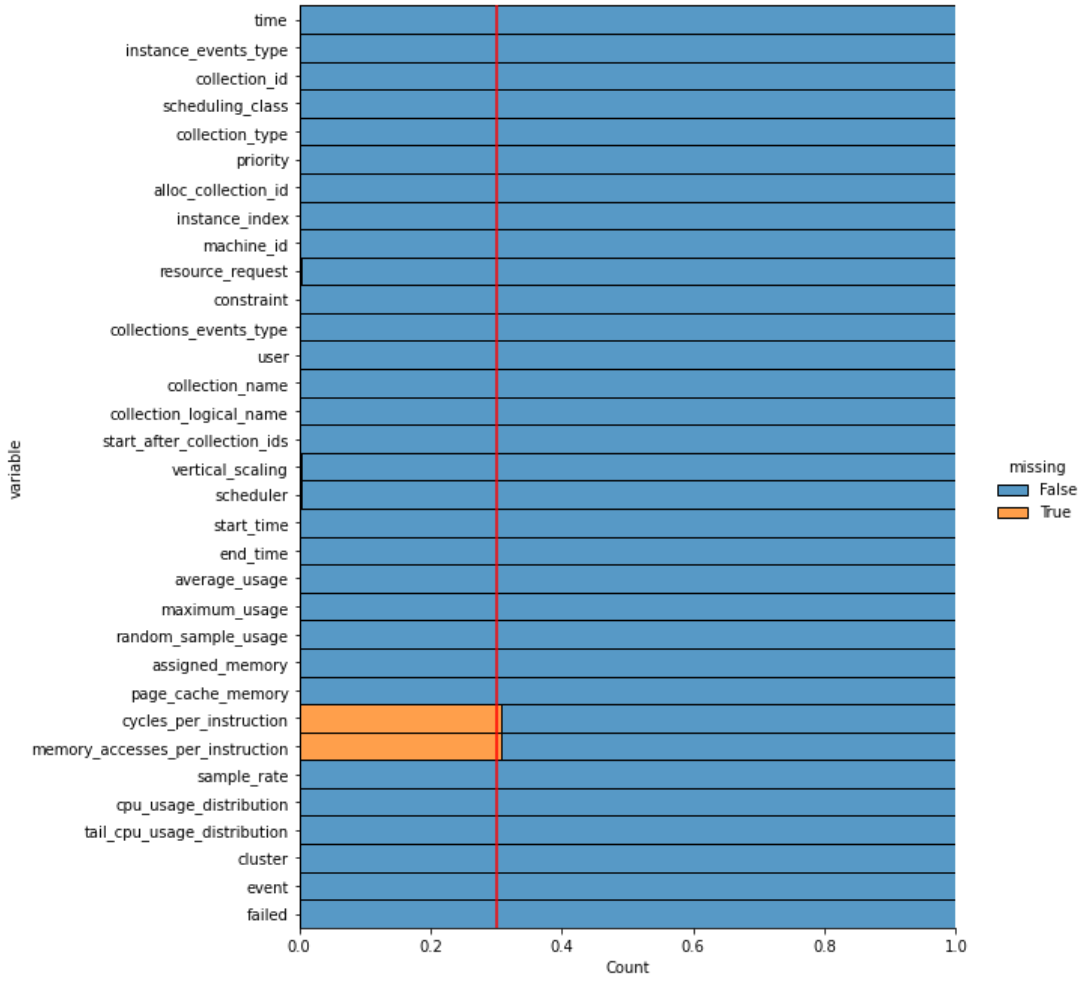

Figure 3. Instances and missing values

Assigned memory is the mean memory boundary (upper bound) for this instance that is provided by the Borglet to the kernel of the operating system. page cache memory is the mean amount of memory that is being utilized by the Operating system for the instance's file page cache. Sample rate here refers to the quantity of samples that are collected per second while the window is active. The nominal goal frequency is 1 Hz, although depending on the load on the system, this might be much lower. If, for instance, samples are only collected once every two seconds, the sample rate would be half of that.

The usernames included in this trace are those of Google employees and services. It is probable that the identical internal or external service is being used by production tasks when they share the same login. a MapReduce algorithm, For instance, will generate not just a master task but also a worker job. Whenever an unique program executes numerous tasks, those jobs will nearly always run as though they were provided by the similar user. This is because a single program acts as a spawning point for all of those jobs.

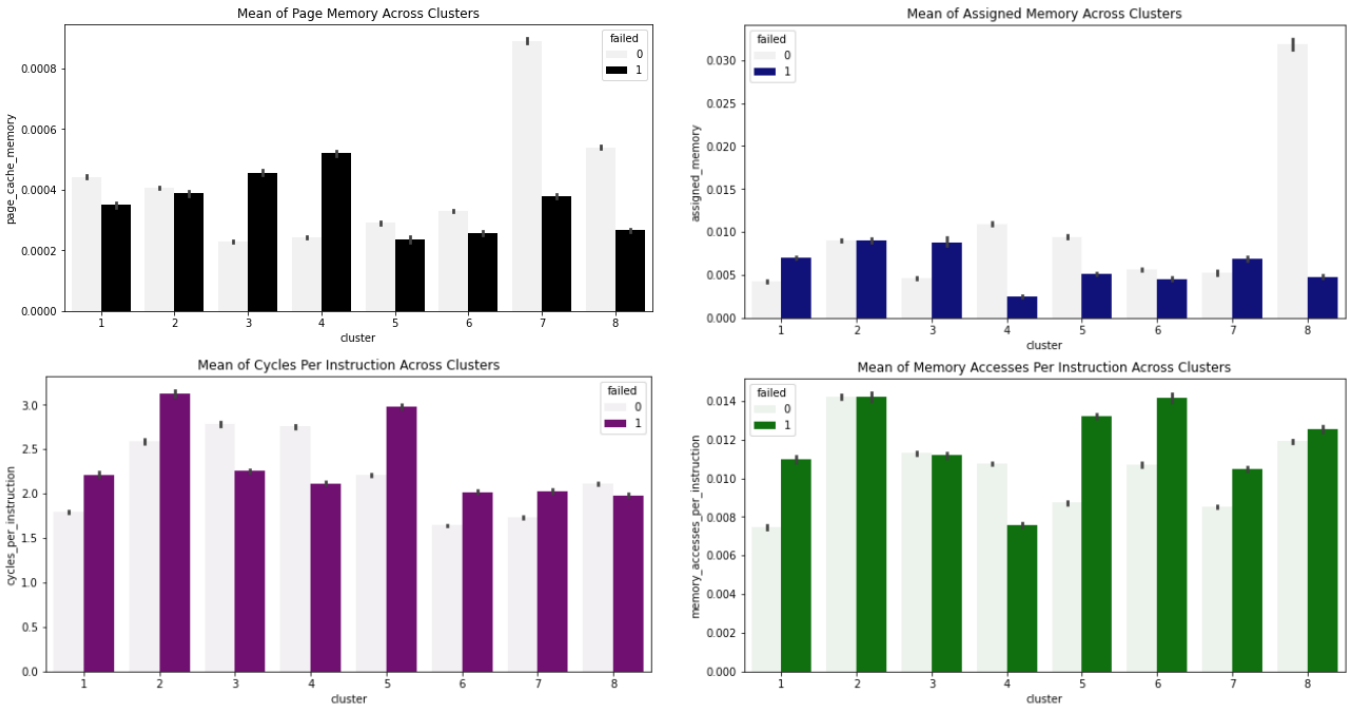

Figure 4. Clusters distribution

It is possible that a task has been submitted, that it has been scheduled and is now ready to be executed, or that its resource demands have been changed. There is a value associated with each collection as well as instance (or thing) occurrence that represents the kind of event. This sort of occurrence always makes it possible to identify the condition that the object will be in after it has occurred. In the case of deaths, the event kind also includes information regarding the circumstances surrounding the passing. The following is a list of the names of the events (transitions): SUBMIT: the cluster manager has been given anything as a submission [29]. QUEUE: an object is placed in a waiting position, often known as a queue, till the scheduler gets prepared to take action on it. ENABLE: a thing has become available to be scheduled; the scheduler will make every effort to position the object as soon as it is capable to. SCHEDULE: the machine was programmed to have something scheduled on it [30]. Because of the time it takes to send the code, etc., it might not begin operating right away. When one of a collection's instances is scheduled for execution on a computer for the very first time, this condition is met. EVICT: a thing has been descheduled due to a thing with a higher priority, since the scheduler overextended and the actual demand surpassed the machine ability, since the machine upon which it had been scheduled had become unusable (for example, because it was taken offline for maintenance), or because the thing's data has been lost for some reason [27]. EVICT: a thing has been descheduled due to a thing with a higher priority (this is very rare).

FAIL: an object has been de-scheduled (or, in unusual situations, stopped to be qualified for scheduling when it was currently on hold) as a result of an error of some sort in a user program, like a segfault, or

an operation to use more memory than it demanded. This could have happened because the user program requested a certain amount of memory. KILL: a thing was terminated by the user or a driver program [31], or the item's parent terminated, or perhaps another thing upon which this job was dependent ended. LOST: a thing has been probably finished, but a record stating it in our source data wasn't there, thus we lost track of when it was finished. This realization was captured in its entirety by the incident in question [32]. UPDATE PENDING is a status that indicates a change has been made to a thing's scheduling class, capacity needs, or limitation while the item was pending to be scheduled.

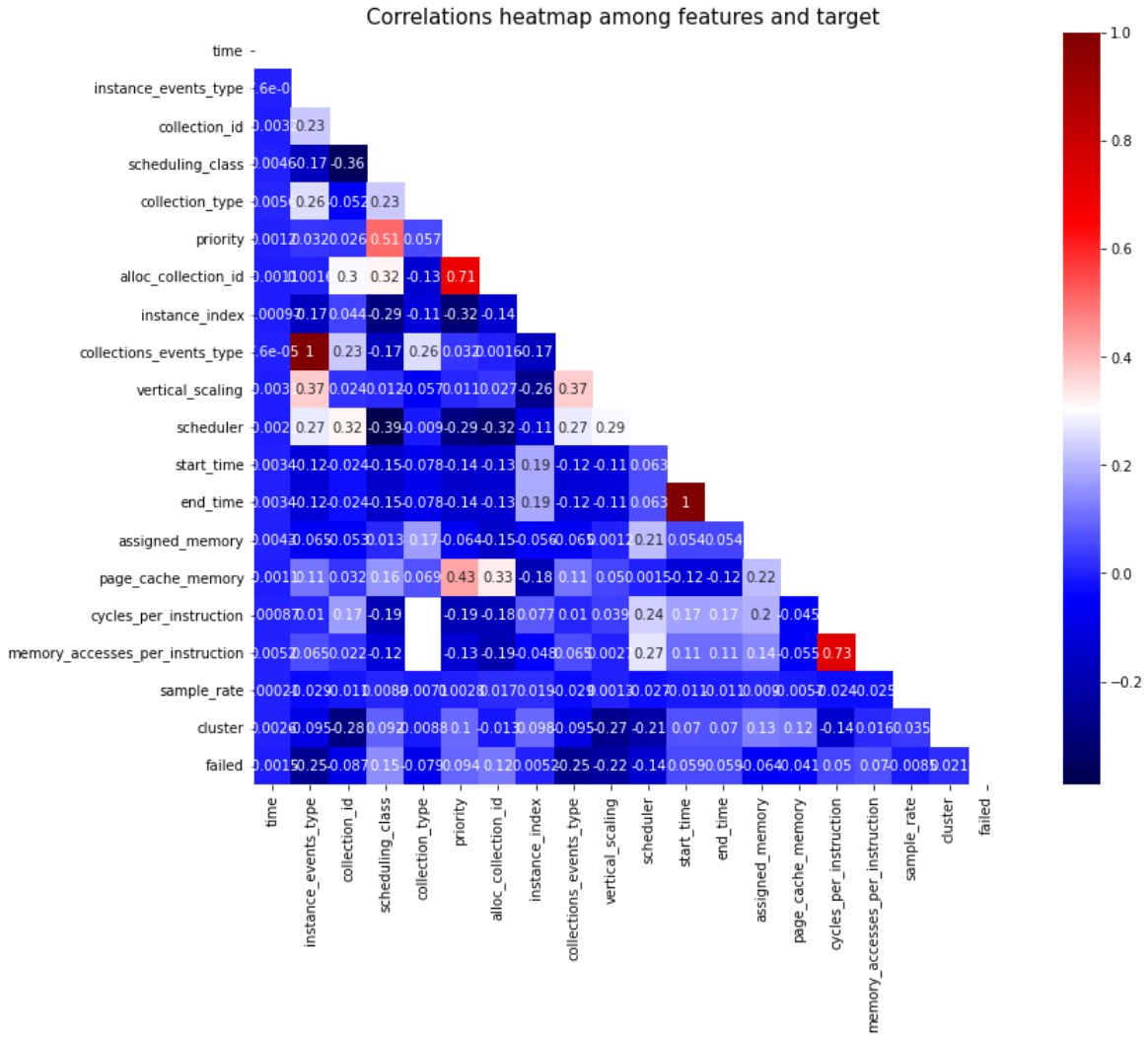

Figure 5. Correlation

Results

Each square represents the level of correlation between the variables given along each axis. The correlation scale ranges from one to one. When the numbers are close to zero, it implies that the two elements do not have a linear connection. The extent to which the intensity of the color fluctuates is the indication of correlation. The nearer the correlation is to one, the stronger the relationship between the two variables. This indicates that when one factor rises, so does the other, and the nearer the correlation is to one, the stronger the link. A correlation closer to -1 is similar to the preceding one, except instead

of both factors increasing as the other increases, one variable decreases. Because these squares are linking each element to itself, the diagonals are all dark red. This implies that the connection is precise. The magnitude of the number and the intensity of the color indicate the level of correlation between 2 variables. Because the same two elements are clustered jointly in every of those squares, the graph is also symmetrical along the diagonal.

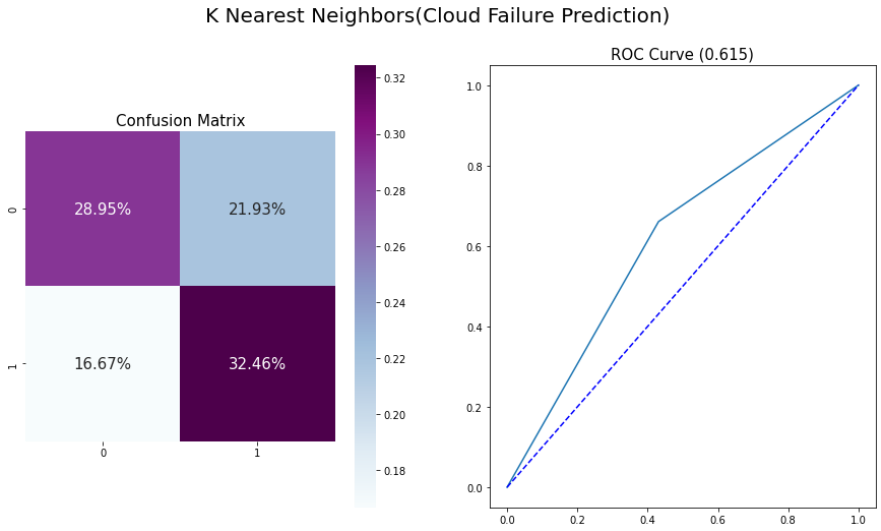

Figure 6. KNN performance for prediction cloud failure

In case of fault prediction, the confusion matrix can be used as a measuring tool in the process of correcting categorization challenges. It may be utilized in cases where there are two class classification algorithms as well as scenarios where there are more than two class classification algorithms. Confusion matrices include counts from both the predicted and actual values. The "TN" output, which refers to "True Negative," displays the number of correctly detected negative situations. Similarly, "TP" is an acronym for "True Positive," which reflects the number of correctly detected positive instances. The term "FN" denotes a false negative value, which is the number of exact positive cases cases classified as negative; the term "FP" denotes a false positive value, that is the number of true negative examples classified as positive; and "FP" denotes a value that shows the number of actual positive cases classified as negative. In the categorization tasks, one of the most often employed criteria is accuracy. We selected to utilize one model as the base for our study if it recognized a TP as well as a TN more correctly than any other model. A

During cloud fault prediction, ROC curve can show the efficiency of a classification method over all categorization levels. This curve displays two parameters: the proportion of True Positives and the percentage of False Positives. Drawing the genuine positive rate on one side and the false positive rate on the opposite yields a ROC curve. The true positive rate (TP/(TP + FN)) is the proportion of positive observations that were correctly predicted to be positive out of the overall frequency of positive observations. Similarly, the false positive ratio (FP/(TN + FP)) is defined as the proportion of data that is incorrectly predicted to be positive from all negative data. A discrete classification which only supplies the anticipated class produces a single point on the ROC space. However, we may build a curve by altering the score threshold for probabilistic classifiers that offer a likelihood or score indicating the amount to which an item corresponds to one group rather than another.

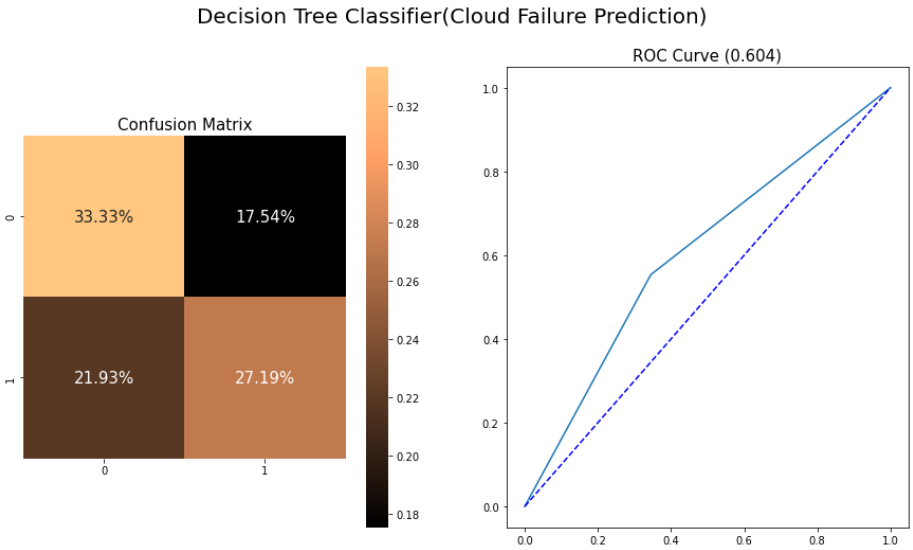

Figure 7. Decision tree performance for prediction cloud failure

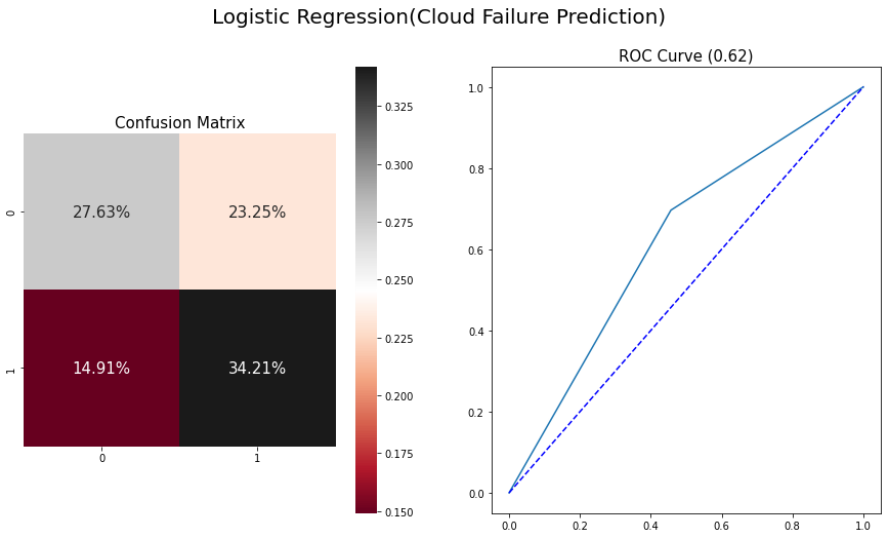

Figure 6. Logistic regression performance for prediction cloud failure

The ROC curve depicts the compromise that must be considered between sensitivity (represented by TPR) and specificity (represented by 1 minus FPR). Classifiers that create curves that are closer to the top left corner may yield better results. It is realistic to expect a random classifier to offer results with points on the diagonal, where FPR = TPR. The accuracy score reduces when the curve near the diagonal line at a degree of 45 degrees or less.

Performance summary for cloud fault predictor classifiers

The training accuracy for KNN is: 75.0733137829912 %

The testing accuracy for KNN is: 61.40350877192983 %

The F1 score for K Nearest Neighbors is: 0.6271186440677966

The training accuracy for logistic regression is: 67.88856304985337 %

The testing accuracy for logistic regression is: 61.8421052631579 %

The F1 score for Logistic Regression is: 0.6419753086419753

The training accuracy for Decision Tree Classifier is: 85.77712609970675 %

The testing accuracy for Decision Tree Classifier is: 60.526315789473685 %

The F1 score for Decision Tree Classifier is: 0.5794392523364487

The training accuracy for SVC is: 68.62170087976538 %

The testing accuracy for SVC is: 61.40350877192983 %

The F1 score for Support Vector Classifier is: 0.6271186440677966

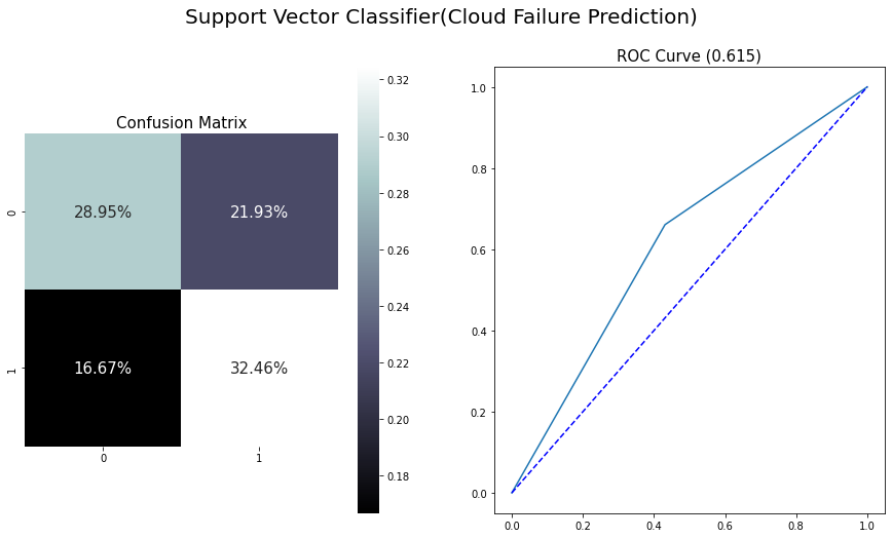

Figure 9. SVM performance for prediction cloud failure

Conclusion

As the discipline of cloud resource management has grown in prominence over the last decade, so too has the importance of failure prediction cloud platforms. This study makes an effort to utilize machine learning algorithm to assist in prediction cloud failure. Although the result could not receive impressive accuracy, it sheds light on the utilization of machine learning in proactive fault tolerance policy. In order to provide satisfactory service to both consumers and companies, faults that occur in cloud data centers and servers need to be identified and forecast accurately so that measures may be put into place to endure the failures that do take place. A problem that originates in one of the housed datacenters may spread to the other datacenters, which would make the situation much more difficult. One can forecast a failure that will propagate across the cloud computing network and then deploy procedures to proactively deal with it. This will avoid scenarios like the one described above from occurring. One method for predicting failures is to instruct a computer to do so on the grounds of communications or logs that are sent between the different cloud-based components. This is one technique to do this. During the training process, the machine has the ability to recognize certain message patterns that are associated with the failure of data centers. After some time has passed, the machine may be used to determine whether or not a certain collection of message logs adhere to such patterns.

References

[1] A. Bala and I. Chana, "Fault tolerance-challenges, techniques and implementation in cloud computing," *International Journal of Computer Science Issues*, 2012.

[2] M. Hasan and M. S. Goraya, "Fault tolerance in cloud computing environment: A systematic survey," *Comput. Ind.*, vol. 99, pp. 156–172, Aug. 2018.

[3] M. Nazari Cheraghlou, A. Khadem-Zadeh, and M. Haghparast, "A survey of fault tolerance architecture in cloud computing," *Journal of Network and Computer Applications*, vol. 61, pp. 81–92, Feb. 2016.

[4] Z. Amin, H. Singh, and N. Sethi, "Review on fault tolerance techniques in cloud computing," *Int. J. Comput. Appl. Technol.*, 2015.

[5] A. Bakr, A. A. El-Aziz, and H. A. Hefny, "A survey on mitigation techniques against ddos attacks on cloud computing architecture," *International Journal of Advanced Science and Technology*, vol. 28, no. 12, pp. 187–200, 2019.

[6] Q. Yan and F. R. Yu, "Distributed denial of service attacks in software-defined networking with cloud computing," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 52–59, Apr. 2015.

[7] A. Waqas, Z. M. Yusof, and A. Shah, "Fault tolerant cloud auditing," *2013 5th International*, 2013.

[8] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, "Consistency-based service level agreements for cloud storage," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, Farminton, Pennsylvania, 2013, pp. 309–324.

[9] K. Lu *et al.*, "Fault-tolerant Service Level Agreement lifecycle management in clouds using actor system," *Future Gener. Comput. Syst.*, vol. 54, pp. 247–259, Jan. 2016.

[10] R. Jhawar and V. Piuri, "Chapter 9 - Fault Tolerance and Resilience in Cloud Computing Environments," in *Computer and Information Security Handbook (Third Edition)*, J. R. Vacca, Ed. Boston: Morgan Kaufmann, 2017, pp. 165–181.

[11] B. B. Nandi, H. S. Paul, A. Banerjee, and S. C. Ghosh, "Fault Tolerance as a Service," in *2013 IEEE Sixth International Conference on Cloud Computing*, 2013, pp. 446–453.

[12] V. Bandari, "The Adoption Of Next Generation Computing Architectures: A Meta Learning On The Adoption Of Fog, Mobile Edge, Serverless, And SoftwareDefined Computing," *ssraml*, vol. 2, no. 2, pp. 1–15, 2019.

[13] R. Tripathi, S. Vignesh, and V. Tamarapalli, "Cost-aware capacity provisioning for fault-tolerant geo-distributed data centers," in *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, 2016, pp. 1–8.

[14] Patra, Singh, and Singh, "Fault tolerance techniques and comparative implementation in cloud computing," *Ann. Math. Inform.*, 2013.
[15] S. Di, Y. Robert, F. Vivien, D. Kondo, C.-L. Wang, and F. Cappello, "Optimization of cloud task processing with checkpoint-restart mechanism," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Denver, Colorado, 2013, pp. 1–12.
[16] A. M. Sampaio and J. G. Barbosa, "A comparative cost analysis of fault-tolerance mechanisms for availability on the cloud," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 315–323, Sep. 2018.
[17] C. Colman-Meixner, C. Develder, M. Tornatore, and B. Mukherjee, "A Survey on Resiliency Techniques in Cloud Computing Infrastructures and Applications," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2244–2281, thirdquarter 2016.
[18] Y. Sharma, B. Javadi, W. Si, and D. Sun, "Reliability and energy efficiency in cloud computing systems: Survey and taxonomy," *Journal of Network and Computer Applications*, vol. 74, pp. 66–85, Oct. 2016.
[19] L. Peterson, "K-nearest neighbor," *Scholarpedia J.*, vol. 4, no. 2, p. 1883, 2009.
[20] Z. Zhang, "Introduction to machine learning: k-nearest neighbors," *Annals of Translational Medicine*, vol. 4, no. 11, Jun. 2016.
[21] Urso, Fiannaca, La Rosa, and Ravì, "Data mining: Prediction methods," . *Comput. Biol. ABC …*, 2018.
[22] P. Langley, *Elements in Machine Learning*. Oxford, England: Morgan Kaufmann, 1995.
[23] M. Paluszek and S. Thomas, *MATLAB Machine Learning*, 1st ed. Berlin, Germany: APress, 2016.
[24] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. Syst. Man Cybern.*, vol. 21, no. 3, pp. 660–674, May 1991.
[25] Y.-Y. Song and Y. Lu, "Decision tree methods: applications for classification and prediction," *Shanghai Arch Psychiatry*, vol. 27, no. 2, pp. 130–135, Apr. 2015.
[26] Sharma and Kumar, "A survey on decision tree algorithms of classification in data mining," *J. Sci. Res. Chulalongkorn Univ.*, 2016.
[27] M. S. Jassas and Q. H. Mahmoud, "Failure characterization and prediction of scheduling jobs in google cluster traces," in *2019 IEEE 10th GCC Conference & Exhibition (GCC)*, 2019, pp. 1–7.
[28] L. Ruan, X. Xu, L. Xiao, F. Yuan, and Y. Li, "A comparative study of large-scale cluster workload traces via multiview analysis," *2019 IEEE 21st*, 2019.
[29] B. Liu, Y. Lin, and Y. Chen, "Quantitative workload analysis and prediction using Google cluster traces," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 935–940.
[30] A. Rosà, L. Y. Chen, R. Birke, and W. Binder, "Demystifying Casualties of Evictions in Big Data Priority Scheduling," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 4, pp. 12–21, Jun. 2015.
[31] Z. Dong, W. Zhuang, and R. Rojas-Cessa, "Energy-aware scheduling schemes for cloud data centers on Google trace data," in *2014 IEEE Online Conference on Green Communications (OnlineGreenComm)*, 2014, pp. 1–6.
[32] M. Soualhia, F. Khomh, and S. Tahar, "Predicting Scheduling Failures in the Cloud: A Case Study with Google Clusters and Hadoop on Amazon EMR," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, 2015, pp. 58–65.