Journal of Theoretical and Applied Information Technology
31st May 2023. Vol.101. No 10
© 2023 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

HYBRIDML: FAULTY NODE DETECTION USING HYBRID LEARNING MODEL FOR DISTRIBUTED DATA CENTRES

ATUL V. DUSANE[1], DR. KRISHNAKANT. P. ADHIYA[2]

[1] Ph.D. Research Scholar, Department of Computer Engineering, SSBT's College of Engineering & Technology Jalgaon.MH, India.

[2] Professor, Department of Computer Engineering, SSBT's College of Engineering & Technology Jalgaon.MH, India.

E-mail: [1]atuld.1987@gmail.com, [2]adhiyakp@gmail.com

ABSTRACT

The distributed systems are very effective when it deals with massive data processing. Nowadays, entire world generates high-dimensional data such as audio, video, image etc. To process such extensive data at a minimum is hard for a stand-alone machine, and this is a big challenge for the computer system to evaluate such data. The distributed framework is the solution for the process of such extensive data. Still, during the execution, some faulty or straggler nodes can increase the overall computation time to process data. However, to detect such straggler nodes, from large distributed systems are mandatory before assigning jobs to VM. Early identification of such faulty node can future save the overall computation time. In this paper, we proposed a hybrid machine learning model for detecting faulty nodes in large distributed machines using collaboration of reinforcement and supervised machine learning. The large Virtual Machine (VM's) log data has been collected from the distributed environment and proceeded with reinforcement learning algorithm for module training and supervised machine learning for module testing. According to extracted features, reinforcement learning encompasses an activation function that generates the label for the respective node, whether healthy or faulty. In the testing phase, the natural world VM's log data has been collected and evaluated with supervised machine learning classifiers. Several machine learning classification algorithms have evaluated and acquired the results. The SVM provides higher accuracy over the other machine learning classifiers with our reinforcement learning model.

Keywords: *Supervised Machine Learning, Classification, Faulty Node, Straggler Node Detection, Distributed System*

1. INTRODUCTION

These days, emerging trends, such as IoT, cloud, health care, VANET, intelligent city applications etc., generate a high volume of data. To process such data in a required large processing model or high-performance computation[1]. Because of this ongoing rise in data velocity refers to the speed, computing capabilities systems may be utilized [2] [4], which perpetuates the need for customizable, computer-controlled scheduling [8]. This challenge is the primary subject of this work, which studies several solutions with the specific goal of reducing straggler tasks. Stragglers are tasks within a job that take much longer to execute than other tasks, and they can produce a serious increment in response time because of the need to synchronize the outputs of the tasks with one another. These challenges can be avoided by carefully planning the order in which tasks are performed. The existence of them may result in something known as the Long Tail Phenomenon [5]. To be more specific, the Long Tail Problem happens when the amount of time needed to complete a certain project is considerably altered in an unfavorable manner by a small number of prone to failures activities. Any gathered technology that performs jobs comprised of several tasks may be susceptible to the phenomenon known as task stragglers. Examples of such systems include Google's Software tool [6] and the Hadoop architecture [7], both of which provide methods for the avoidance of stragglers as standards [1], [8], and [9].

Both the MapReduce and the Hadoop frameworks enable a system to scale to enormous cheap commodity machines. By the concepts outlined in

IBM's autonomous model [10], [11], the execution of activities in parallel not only enhances the pace at which they are carried out but also deals automatically and independently with any problems that may occur. However, stragglers may still emerge due to software or hardware problems because autonomic approaches are often delayed in managing failures. This can result in extended downtime for devices with limited resources [1]. These contribute to unanticipated delays in the work execution owing to the financial constraints or the loss of data. They cause such jobs to hog resources, leading to more extraordinary reaction times in the case of non-primitive execution.

Therefore, effective methods are essential to reduce stragglers to avoid excessive response times and breaches of SLA agreements. Now we will talk about the many mistakes that result in straggler's chores. While carrying out jobs, there is a possibility of encountering either a task failure or a node failure. The former scenario occurs when a job's tasks fail for various reasons, including software and hardware [12]. This latter scenario plays out when one of the components of a particular node, which is responsible for carrying out the task of the job, fails [1]. Several problems at the operating system (OS) or hardware level might be to blame. As an illustration of a strategy for straggler mitigation, MapReduce tries to alleviate task failures by relaunching the job after it has failed [13]. This is an example of a straggler mitigation approach. If a node fails, MapReduce will carry out once again all of the original activities planned to be carried out on that node. In terms of system crashes, when the effectiveness of a node deteriorates, either because of a fault in the operating system or the hardware or when the node fails spectacularly. The execution time of a particular task, known as a "straggler," can become excessively long, which forces any other work activities that rely heavily on it to wait for it to finish [14].

For the job to be finished at the level of the work, each of the tasks that make up the job must be finished. If one straggling task affects other related tasks from being completed, the work will not be finished until all of the straggling tasks have been finished [15]. In addition, straggler jobs can potentially delay the completion of other activities dependent on their output, causing those other processes to use extra resources and further hinder the performance of the computer system. Stragglers have an impact not just on performance but also on the costs of deployment. The problem of straggler jobs, which may cause a delayed response or waste resources, is a concern faced by well-known cloud service providers, such as Amazon, Google, Netflix, and Apple. This necessitates an unnecessary scaling-up of the network infrastructures, which increases deployment costs [14], [16]. The performance of cloud services is also impacted by instances of high latency that are referred to be "tail-tolerant" or "latency-tail-tolerant" [17]. Jobs that are tolerant of latency have a negative impact on resource usage and a positive one on energy consumption. Investigations such as [1, [2], [5], [6], [10], [12], and [18] reveal that resource contention is the primary cause for stragglers, which occurs when various tasks are waiting for shared infrastructure. This is shown to be the case by the findings. Programmes on different nodes can compete for globally interconnected resources [17]. Previous research [19] focuses on resolving the issue of straggler tasks by identifying and resolving which activities are stragglers only when the jobs have been completed. This approach was used to solve the problem of missing value tasks. The term "straggler mitigation" involves the protection of any influence that straggler tasks may have on the quality of service or the service level agreement.

2. LITERATURE REVIEW

In this section, we demonstrate various state-of-art system developers by previous researchers. In Google cluster use traces, Mesbahi et al. [20] provide a trustworthiness analysis and a Markov model. They used different physical machine probability of failure and attributes, including steady-state unavailability, mean time to rebuild and between failures to study the dependability of Google cluster traces. In a case study on Amazon and Search engine Cluster Trace, Ruan et al. [21] used a multi-view technique to compare two cloud workloads. Using the Google cluster workload, Ahmed et al. [22] have discovered the density function again for time to restore and the probability of failure for cloud servers. Unsupervised machine learning has been used to describe cloud approaches based on job and activity events [23,24]. Di et al. used a K-means clustering approach with an appropriate number of sets to identify cloud-based applications based on task occurrences and resource use. The distribution of applications in the K-means classification sets is analogous to the Pareto distribution. Resources and workload patterns were studied statistically by Alam et al. [24]. Even though multiple prior studies have examined Google clustering traces for required to fill, the significant contributions of this research are the grouping of Google required to fill and the categorization of jobs using K-means clusters.

Four new traces exist, two from private clusters and two from high-performance computing centres (HPCs). According to their results, there is a correlation between the data analysis jobs performed in private clusters and those performed in HPC clusters. New results should be evaluated in the context of previous evidence, as this observation indicates. LANL's high-performance computer clusters and Two Sigma's private cloud each contributed additional traces to the analysis [26]. In [27], we looked at things like memory use, CPU speed, and available storage space related to workload. Failed jobs tend to have many aspects in common with those that succeed. In addition, there is significant evidence that cloud resources were used for either cancelled or failed activities. A tiny fraction of rejected tasks were resubmitted numerous times to finish them successfully. As a result, these failed operations were classified as "killed" jobs since they used up many resources. None of the three-class activities worked out. The scheduling category and failure go hand in hand, as shown in this case.

Zhao et al. [28] use an entirely new technique for disc failure detection than earlier studies. They employ distinct properties measured at consecutive time intervals for a disc drive as time series, and they apply HMM and Embedded Semi-Markov Model (HSMM) to simulate such moving average to detect "failed" discs from "good" discs. Auto-scaling services may be constructed using various CPU prediction methods, such as Linear Regression, Due To attachment and Auto Regression Incorporated Moving Averages (ARIMA) (ARIMA). Moreover, pattern identification and a state-driven approach to estimate occupations by Gong et al. [30] construct the workload forecasting model called Predictive Dynamic resource Scalability. Signal processing methods are utilized to identify whether or not the CPU in a virtual machine exhibits recurring activity patterns. In this case, the recurring patterns are used to predict future workloads; if the response is no, PRESS uses a statistical state-driven approach. Predicting future demand is done using a discrete-time Dynamical system.

Samak et al. [35] used the Bayesian Classification method to forecast task failure based on the logs of science processes. Then they demonstrated that a job projected to fail might be successfully scheduled to another available resource in some instances. Proactive fault tolerance strategies were presented by Bala and Chana [36], utilizing analysis of the data and machine learning techniques to detect job failures. In this way, they use their method throughout the execution of the applications before the failure.

Hongyan et al. evaluated machine learning classifiers to predict work failures [37]. They tested four algorithms: RF, KNN, KDT, and LR, and compared their results. It is necessary to utilize the OpenCloud dataset to verify the classifier's accuracy. It was shown that Sun et al. [38] could accurately forecast software failures using a deep learning model. Additionally, they devised a way to manufacture new failure data by producing fresh samples. For scientific applications, Padmakumari and Umamakeswari [39] used a variety of machine learning classifiers to forecast task failure. A fictitious collection of data was used to train and evaluate classifiers. This study's findings show that the NB classifier is accurate (up to 94.9 per cent). Workplace failures can be predicted using deep learning, according to Gao et al. [40]. Many layers were used in a model known as Bidirectional Long Short Term Memory (Bi-LSTM). The authors use static and dynamic features to model validation data sets. Bi-LSTM forecasted task failure with an effectiveness of up to 93 per cent and job failure with an accuracy of up to 87 per cent, according to the data. The failure of a cloud application may be attributed to several factors, including the following: The Sequence-to-sequence model was used to anticipate the application's end state. According to these findings, LSTM is up to 87% accurate. In our prior work [41], we suggested failure forecasting models and used numerous feature selection strategies to increase the model's accuracy. Prediction models developed by us surpassed those developed by others in earlier research.

The above survey describes detecting faulty or straggler nodes from distributed VMs. However Most researchers have used the statical analysis method, and some are supervised machine learning and deep learning classifiers. The major problem of this system is the low detection rate for accurate detection of faulty nodes due to insufficient training model or features required for robust module building.

3. PROPSOED SYSTEM DESIGN

The below Figure 1 describes a faulty node detection using proposed hybrid model, with reinforcement learning and supervised learning model. Through the cooperation of a number of different hybrids learning algorithms, this body of work proved the ability to identify a defective node in a decentralized system. It has been gathered from the log data of dispersed virtual machines, and several characteristics, including system memory, CPU load, activity RAC, etc., have been retrieved from the data. The fragment that was eliminated was used as input into the classifier that was being trained for a particular machine learning method. The application of categorization has been implemented inside the WEKA 3.7 framework. In order to verify the work that was presented, we conduct an analysis of almost six different machine learning methods and compile the likely outcomes from each classifier.

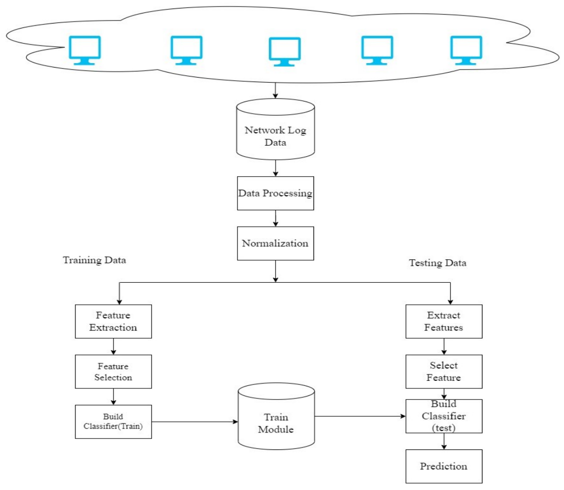

Figure 1 :Proposed System Architecture For Faulty Node Detection In Distributed Systems

The whole process results in the identification of straggler nodes as the output, and the system then automatically blocks any such nodes that are found. The following step-by-step instructions should be followed exactly for the complete execution. Before distributing any work to any nodes, we first retrieve the data proximity characteristics, CPU load, and memory load from all of the virtual machines. Then, we send the different jobs to every other node to be processed. This information is taken into account for load data in order to locate the straggler throughout the complete operation.

When it came to identification and tracking, all we did was apply HML as a simple supervised classification algorithm. Following this, a technique based on machine learning is used to train the classifiers. At the beginning, information along with its classifications is delivered. When trying to determine whether someone is lying about their identity on social media, the HML technique uses a number of different decision trees, each of which is generated by randomly selecting a feature from a set of features. Then, the HML technique selects the decision tree whose outcomes are most common as the final result.

4. IMPLEMENTATION DETAILS

The below algorithm technique presents a solution that gets around the problem of straggler nodes being predicted, and it does so use the hybrid supervised classification method. The total values for the retrieved parameters are shown by the proposed hybrid technique. These values are calculated using a variety of machine learning algorithms. The following methodology was used in order to construct the machine performance report in accordance with the combination reinforcement learning based prediction method.

Input: Normalized training dataset *Train_Data[]*, Normalized testing dataset *Test_Data[]*, defined threshold *qTh*
Output: Result set as output with *{Predicted_class, weight}*

Step 1: Read all test data from *Test_Data[]* using below function for validating to training rules, the data is normalized and transformed according to algorithms requirements

$$test\_Feature(data) = \sum_{m=1}^{n} (.Attribute\_Set[A[m] \ldots \ldots A[n] \leftarrow Test\_Data)$$

Step 2: select the features from extracted attributes set of $test\_Feature(data)$ and generate feature map using below function.

$$Test\_FeatureMap\ [t \ldots \ldots n] = \sum_{x=1}^{n} (t) \leftarrow test\_Feature(x)$$

$Test\_FeatureMap\ [x]$ are the selected features in pooling layer. The convolutional layer extracts the features from input and passes to pooling layer and those selected features are stored in *Test_FeatureMap*

Step 3: Now read entire taring dataset to build the hidden layer for classification of entire test data in sense layer,

$$train\_Feature(data) = \sum_{m=1}^{n} (.Attribute\_Set[A[m] \ldots \ldots A[n] \leftarrow Train\_Data)$$

Step 4: Generate the training map using below function from input dataset

$$Train\_FeatureMap\ [t \ldots \ldots n] = \sum_{x=1}^{n} (t) \leftarrow train\_Feature(x)$$

$Train\_FeatureMap[t]$ is the hidden layer map that generates feature vector for build the hidden layer. That evaluate the entire test instances with train data.

Step 5: After generating the feature map we calculate similarity weight for all instances in dense layer between selected features in pooling layer

$$Gen\_weight = CalcWeight\left(Test\_FeatureMap \,\|\, \sum_{i=1}^{n} Train\_FeatureMap[i]\right)$$

Step 6: Evaluate the current weight with desired threshold

$if(Gen\_weight >= qTh)$

Step 7 : Out_List.add (trainF.class, weight)

Step 8: Go to step 1 and continue when Test_Data== null

Step 9: Return Out_List

This composite classification method receives input in the form of all potential outcomes produced by specified data mining algorithms. These results are then fed as input to the composite classification algorithm. As per the given values of probabilities method in step 5, it creates the exact weight for a particular virtual environment, and on the basis of that, and we may estimate the potential of straggler for VM.

5. RESULTS AND DISCUSSION

The expansive Java platform with JDK 1.8 and the Weka 3.7 framework were used to carried out the development of the proposed system. In this extensive experimental analysis, the different types of machine learning approaches were used to determine the likelihood of straggler identification. The system provides a description of five assessments that have been carried out using three distinct forms of cross validation. The SVM offers the best performance, with a detection performance of 96% when subjected to 20-fold classification model.

Table 1: Dataset Description

Name	Description	Missing values
Log-1	Log data of distributed VM's with 10 nodes	Yes
Log-2	Log data of distributed VM's with 25 nodes	Yes
Log-3	Log data of distributed VM's with 50 nodes	Yes
Log-4	Log data of distributed VM's with 100 nodes	Yes

The Table 1, demonstrates dataset used for detection and prediction of straggler node. The four datasets have used with different size of virtual machines.

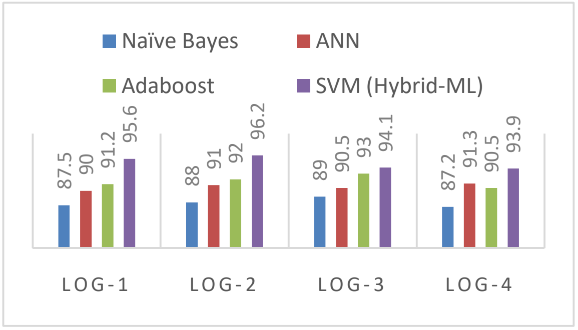

Figure 2: Comparative Analysis Of Proposed Model With Various Log Dataset For Faulty Node Detection

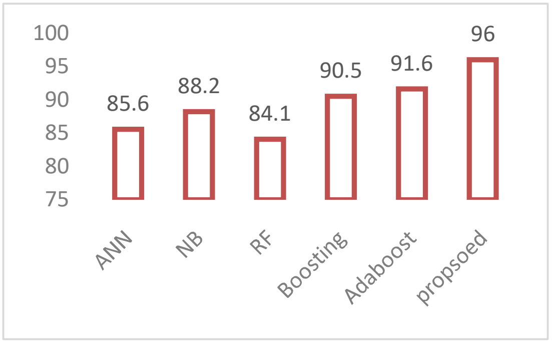

Figure 3: Accuracy Of Straggler Node Identification By The Use Of Machine Learning And Hybrid Classification

As can be seen in Figure 3, five different categorization strategies were used in an effort to locate straggler nodes on the decentralized log collection. Out of all the classifications, the SVM has the highest accuracy, whilst the RF classification has the lowest accuracy. In order to evaluate the effectiveness of such categorization strategies, we made use of another database to generate test cases based on a variety of log information sources. The performance study was conducted out on a number of different existing systems, each of which used equivalent machine learning methodologies to construct analogous tactics. A further experiment including a large number of data samples was carried out, and the relevant evaluations of sensitivity and specificity were carried out. Our hybrid classification system has a detection accuracy of 96%, which is often the highest among other classifiers.

6. CONCLUSION

This work describes a faulty node detection and prediction using hybrid machine learning model. We've shown that our approach outperforms some of the more well-known alternatives. With just a third of the learning algorithm, our formulation is more accurate and generalizable than existing techniques for active learning requiring little. As a result, the issue of class disparity is dealt with progressively. For straggler identification, our technique is more suited since it is able to accurately represent the straggler nodes distribution. The proposed models archive 96% average accuracy for various log dataset with hybrid SVM. Identifying stragglers in operation with great reliability is possible because to this performance. Rather than being limited to big data computing architectures, the suggested framework may be used to a wide range of workloads (e.g., across several data centers). Additional information may be gleaned from the node and task usage resources mentioned in this framework.

7. FUTURE WORK

To develop various deep learning models for effective detection and prediction of straggler nodes in large distributed environments.

REFERENCES

[1] S. S. Gill, X. Ouyang, and P. Garraghan, "Tails in the cloud: a survey and taxonomy of straggler management within large-scale cloud data centres," The Journal of Supercomputing, pp. 1–40, 2020.

[2] H. Xu and W. C. Lau, "Optimization for speculative execution in big data processing clusters," IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 2, pp. 530–545, 2016.

[3] M. Liaqat, A. Naveed, R. L. Ali, J. Shuja, and K.-M. Ko, "Characterizing dynamic load balancing in cloud environments using virtual machine deployment models," IEEE Access, vol. 7, pp. 145 767– 145 776, 2019.

[4] S. Mustafa, K. Sattar, J. Shuja, S. Sarwar, T. Maqsood, S. A. Madani, and S. Guizani, "Sla-aware best fit decreasing techniques for workload consolidation in clouds," IEEE Access, vol. 7, pp. 135 256–135 267, 2019.

[5] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," ACM SIGMETRICS Performance Evaluation Review, vol. 43, no. 3, pp. 7–11, 2015.

[6] E. Coppa and I. Finocchi, "On data skewness, stragglers, and mapreduce progress indicators," in Proceedings of the Sixth ACM Symposium on Cloud Computing. ACM, 2015, pp. 139–152.

[7] A. Eldawy and M. F. Mokbel, "Spatialhadoop: A mapreduce framework for spatial data," in 2015 IEEE 31st international conference on Data Engineering. IEEE, 2015, pp. 1352–1363.

[8] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu, "Grass: Trimming stragglers in approximation analytics," in Networked Systems Design and Implementation (NSDI), 2014, pp. 289–302.

[9] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic gradient coding for straggler mitigation in distributed learning," IEEE Journal on Selected Areas in Information Theory, 2020.

[10] S. S. Gill, P. Garraghan, V. Stankovski, G. Casale, R. K. Thulasiram, S. K. Ghosh, K. Ramamohanarao, and R. Buyya, "Holistic resource management for sustainable and reliable cloud computing: An innovative solution to global challenge," Journal of Systems and Software, 2019.

[11] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in 2012 proceedings IEEE Infocom. IEEE, 2012, pp. 945–953.

[12] D. Lindsay, S. S. Gill, and P. Garraghan, "Prism: An experiment framework for straggler analytics in containerized clusters," in Proceedings of the 5th International Workshop on Container Technologies and Container Clouds. ACM, 2019, pp. 13–18.

[13] P. Garraghan, R. Yang, Z. Wen, A. Romanovsky, J. Xu, R. Buyya, and R. Ranjan, "Emergent failures: Rethinking cloud reliability at scale," IEEE Cloud Computing, vol. 5, no. 5, pp. 12–21, 2018.

[14] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in ACM SIGMETRICS Performance Evaluation Review, vol. 42, no. 1. ACM, 2014, pp. 599–600.

[15] U. Kumar and J. Kumar, "A comprehensive review of straggler handling algorithms for mapreduce framework," International Journal of Grid and Distributed Computing, vol. 7, no. 4, pp. 139–148, 2014.

[16] M. F. Aktas, P. Peng, and E. Soljanin, "Effective straggler mitigation: Which clones should attack and when?" ACM

SIGMETRICS Performance Evaluation Review, vol. 45, no. 2, pp. 12–14, 2017.

[17] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz, "Wrangler: Predictable and faster jobs using fewer resources," in Proceedings of the ACM Symposium on Cloud Computing. ACM, 2014, pp. 1–14.

[18] F. Farhat, "Stochastic modeling and optimization of stragglers in mapreduce framework," 2015.

[19] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in Networked Systems Design and Implementation (NSDI), 2012, pp. 2–2

[20] Mesbahi, M.R.; Rahmani, A.M.; Hosseinzadeh, M. Dependability analysis for characterizing Google cluster reliability. Int. J. Commun. Syst. 2019, 32, e4127 .

[21] Ruan, L.; Xu, X.; Xiao, L.; Yuan, F.; Li, Y.; Dai, D. A Comparative Study of Large-Scale Cluster Workload Traces via MultiviewAnalysis. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; pp. 397–404.

[22] Ahmed, K.M.U.; Alvarez, M.; Bollen, M.H. Characterizing failure and repair time of servers in a hyper-scale data center. InProceedings of the 2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe), Hague, The Netherlands, 26–28 October 2020; pp. 660–664.

[23] Di, S.; Kondo, D.; Cappello, F. Characterizing cloud applications on a Google data center. In Proceedings of the 2013 42nd International Conference on Parallel Processing, Lyon, France, 1–4 October 2013; pp. 468–473.

[24] Alam, M.; Shakil, K.A.; Sethi, S. Analysis and clustering of workload in google cluster trace based on resource usage. In Proceedings of the 2016 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC) and 15th International Symposium on Distributed Computing and Applications for Business Engineering (DCABES), Paris, France, 24–26 August 2016; pp. 740–747. doi:10.1109/CSE-EUCDCABES.2016.271.

[25] Amvrosiadis, G.; Park, J.W.; Ganger, G.R.; Gibson, G.A.; Baseman, E.; DeBardeleben, N. On the diversity of cluster workloads and its impact on research results. In Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC 18), Boston, MA, USA, 11–13 July 2018; pp. 533–546.

[26] Amvrosiadis, G.; Kuchnik, M.; Park, J.W.; Cranor, C.; Ganger, G.R.; Moore, E.; DeBardeleben, N. The Atlas cluster trace repository. USENIX Login 2018, 43, 4.

[27] Jassas, M.; Mahmoud, Q.H. Failure Analysis and Characterization of Scheduling Jobs in Google Cluster Trace. In Proceedings of the IECON 2018—44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, USA, 21–23 October 2018; pp. 3102–3107.

[28] Zhao, Y.; Liu, X.; Gan, S.; Zheng, W. Predicting disk failures with HMM-and HSMM-based approaches. In Industrial Conference on Data Mining; Springer: Berlin/Heidelberg, Germany, 2010; pp. 390–404.

[29] Morais, F.J.A.; Brasileiro, F.V.; Lopes, R.V.; Santos, R.A.; Satterfield, W.; Rosa, L. Autoflex: Service agnostic auto-scaling framework for iaas deployment models. In Proceedings of the 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, Delft, The Netherlands, 13–16 May 2013; pp. 42–49.

[30] Gong, Z.; Gu, X.; Wilkes, J. Press: Predictive elastic resource scaling for cloud systems. In Proceedings of the 2010 International Conference on Network and Service Management, Niagara Falls, ON, Canada, 25–29 October 2010; pp. 9–16.

[31] Liang, Y.; Zhang, Y.; Sivasubramaniam, A.; Jette, M.; Sahoo, R. Bluegene/l failure analysis and prediction models. In Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN'06), Philadelphia, PA, USA, 25–28 June 2006; pp. 425–434.

[32] Shetty, J.; Sajjan, R.; Shobha, G. Task Resource Usage Analysis and Failure Prediction in Cloud. In Proceedings of the IEEE 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 10–11 January 2019; pp. 342–348.

[33] Soualhia, M.; Khomh, F.; Tahar, S. Predicting scheduling failures in the cloud: A case study with google clusters and hadoop on amazon EMR. In Proceedings of the 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, New York, NY, USA, 24–26 August 2015; pp. 58–65. doi:10.1109/HPCC-CSSICESS.2015.170.

[34] Alahmad, Y.; Daradkeh, T.; Agarwal, A. Proactive Failure-Aware Task Scheduling Framework for Cloud Computing. IEEE Access 2021. doi:10.1109/TSC.2020.2993728.

[35] Samak, T.; Gunter, D.; Goode, M.; Deelman, E.; Juve, G.; Silva, F.; Vahi, K. Failure analysis of distributed scientific workflows executing in the cloud. In Proceedings of the IEEE 2012 8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualiztion Management (SVM), Las Vegas, NV, USA, 22–26 October 2012; pp. 46–54.

[36] Bala, A.; Chana, I. Intelligent failure prediction models for scientific workflows. Expert Syst. Appl. 2015, 42, 980–989.

[37] Hongyan, T.; Ying, L.; Long, W.; Jing, G.; Zhonghai, W. Predicting misconfiguration-induced unsuccessful executions of jobs in big data system. In Proceedings of the 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Turin, Italy, 4–8 July 2017; Volume 1, pp. 772–777.

[38] Sun, Y.; Xu, L.; Li, Y.; Guo, L.; Ma, Z.; Wang, Y. Utilizing Deep Architecture Networks of VAE in Software Fault Prediction. In Proceedings of the 2018 IEEE International Conference on Parallel & Distributed Processing with Applications), Melbourne, VIC, Australia, 11–13 December 2018; pp. 870–877.

[39] Padmakumari, P.; Umamakeswari, A. Task failure prediction using combine bagging ensemble (CBE) classification in cloudworkflow. Wirel. Pers. Commun. 2019, 107, 23–40.

[40] Gao, J.; Wang, H.; Shen, H. Task failure prediction in cloud data centers using deep learning. IEEE Trans. Serv. Comput. 2020, 9, 106152–106168. https://doi.org/10.1109/TSC.2020.2993728.

[41] Jassas, M.S.; Mahmoud, Q.H. Evaluation of a Failure Prediction Model for Large Scale Cloud Applications. In Canadian Conference on Artificial Intelligence; Springer: Berlin/Heidelberg, Germany, 2020; pp. 321–327