Journal of Information Systems Engineering and Management
2025, 10(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Mitigation and Detection of Faulty Nodes in Multinode Hadoop Cluster using Hybrid Machine Learning Techniques

Atul V. Dusane[1], Chaitali G. Patil[2], Dr. Shivganga. C. Maindargi[3], Dr Suman Kumar Swarnkar[4], Dattatray G Takale[5]
[1]SVKM's NMIMS,Shirpur, Dhule, Maharashtra, India atuld.1987@gmail.com
[2]SVKM's NMIMS, Shirpur, Dhule, Maharashtra, India chaitalipatil01@gmail.com
[3](Assistant Professor- Management Studies) Bharati Vidyapeeth(Deemed to be) University, Pune
Abhijit Kadam Institute of Management and Social Sciences, Solapur
[4]Dept of Computer science and engineering,
Shri Shankaracharya Institute of Professional Management and Technology Raipur, India Sumanswarnkar17@gmail.com
[5]Assistant Professor, Department of Computer Engineering, Vishwakarma Institute of Information Technology, Pune, Maharashtra, India. dattatraygtakale@gmail.com

ARTICLE INFO

Received: 10 Oct 2024

Revised: 09 Dec 2024

Accepted: 22 Dec 2024

ABSTRACT

In today's modern significant computational systems, jobs are broken down into several smaller processes that run simultaneously to increase the rate at which jobs are completed and lower the amount of energy that is consumed. However, dealing with straggler processes, which are sluggish running processes that raise the total response time, is a typical performance challenge in these kinds of systems. These kinds of jobs have the potential to have a substantial effect on the Quality of Service (QoS) provided by the system. It is necessary to have automatic straggler identification and mitigation systems that can complete jobs in a shorter amount of time in order to address this problem. Previous work often constructs reactive frameworks, the central emphasis of which is, in order, the identification, followed by the mitigation, of straggler tasks, that ultimately results in delays. Other research make use of prediction-based proactive systems, however they disregard the peculiarities of heterogeneous hosts or dynamic tasks. In this article, Hybrid Machine Learning (HML) is offered as a method that may determine which jobs are likely to be behind schedule and dynamically adjust scheduling in order to obtain faster response times. The method that has been suggested examines all tasks as well as hosts on the basis of the use of compute and network resources, and it is also able to predict and mitigate the effects of expected straggler activities. This speeds up the execution without lowering the quality of service. The proposed HML is evaluated in terms of quality of service factors such as energy usage, processing time, resource contention, and CPU utilisation in comparison to other machine learning methods that already exist, including Support Vector Machine (SVM), ADABOOST, Artificial Neural Network (ANN), Naive Bayes (NB), Decision Tree (DT), and Random Forest (RF). According to the results of several evaluations, the proposed HML cuts down on processing time, resource contention, and energy usage by 13.5%, 11.25%, and 16.75%, correspondingly, when compared to standard machine learning methodologies. The proposed HML has a performance accuracy of 98.1%, making it superior to those other conventional ML methods.

Keywords: Straggler Node, Mitigation, Hadoop, Map-Reduce, HML, RNN

INTRODUCTION

Areas of application of Cloud Data-Centers (CDCs) in areas such as health care, food production, smart cities, weather prediction, and traffic control produce large amounts of data, which are transmitted among multiple devices utilising various types of communication mechanisms [20]. The constant rise in data quantity and velocity may necessitate the utilisation of huge computational systems [21, 22]. This only serves to heighten the already pressing requirement for techniques of intelligent future employment and adaptable, autonomous scheduling. This challenge is the primary subject of this work, which studies several solutions with the specific goal of reducing straggler tasks. Stragglers are activities within a job which take significantly longer to perform than some other processes, and they can result in a significant rise in turnaround time because of the need to synchronize the outputs of the tasks with one

another. Stragglers can be avoided by carefully planning the order in which tasks are performed. The existence of them raises the risk of something known as the "Long Tail Issue."

To be more specific, the Long Tail Problem happens when the amount of time needed to complete a specific project is considerably altered in an un-favorable way by a tiny proportion of straggler activities. Any highly parallelized software that performs jobs comprising of several tasks may be susceptible to the phenomenon known as task stragglers. Cases of such a framework include Google's MapReduce framework and Hadoop's architecture; both of these example demonstrates that methods for the avoidance of stragglers are popular [20, 23]. The system can be scaled up to large clusters of commodity computers with either MapReduce or Hadoop, which both offer this capability. The performance of operations in parallel enhances the speed at which they are carried out and automatically deals with errors in the absence of any interaction from a human, in accordance with the principles outlined in IBM's autonomic paradigm [24]. Furthermore, stragglers could still emerge as a consequence of software or hardware problems due to the fact that autonomic models are frequently delayed in managing failures. This can result in extended periods of downtime for devices that have limited resources [20]. These contribute to unanticipated delays in the task execution because of the lack of resources or the destruction of data, and they lead such jobs to hog resources, which results in longer reaction times in the case of non-preemptive processing. Therefore, effective strategies are essential to reduce the number of stragglers in order to prevent long reaction times. The many kinds of errors that can result in stragglers being assigned jobs are explained in the following paragraphs.

During the process of carrying out jobs, there is the potential for two distinct kinds of errors to take place: task errors and node faults. The earlier occurs when a particular task inside a job fails, which can be caused by a variety of different software and hardware issues [25]. This latter scenario plays out if one of the resources of a particular node, which is responsible for carrying out the task of the job, fails [20]. This could be due to a wide variety of problems at the operating system or hardware level. MapReduce is an implementation of a concept known as straggler mitigation, in which an attempt is made to reduce the number of failed tasks by relaunching the failed task. In the event that one of the nodes in a MapReduce cluster fails, the system will retry all of the tasks that had been planned to be carried out on that node. In aspects of node failures, whenever the effectiveness of a node deteriorates, either because of a fault in the operating system or in the hardware, or whenever the node fails completely, the processing time of a particular task, known as a "straggler," can become excessively long, forcing any other activities that rely on it to wait for it to finish before continuing. At the level of the work, in order for the task to be regarded finished, each of the tasks that make up the job must be finished. If a straggler task stops another sibling activities from being effectively done, the work won't be finished until all of the straggler tasks have been finished. In particular, straggler jobs might cause other tasks that are relied on their output to wait, which causes more resource consumption and has a further negative impact on the efficiency of the computer system.

Stragglers have a negative impact not only on efficiency but also on the costs of deployment. The problem of straggler tasks, which can cause a delayed response or waste resources, is a concern that is faced by well-known providers of cloud services, like Amazon, Google, Netflix, and Apple. This necessitates an unnecessary scaling-up of the cloud infrastructures, which in effect leads to a rise in the expenses associated with deployment. The efficiency of cloud services is also negatively impacted by instances of high latency that are referred to as "tail-tolerant" or "latency-tail-tolerant." Jobs that are tolerant of latency have a negative impact on resource usage and a positive one on energy consumption. Investigations such as [20, 24, 25] reveal that resource contention is the primary cause of stragglers, which occurs when multiple processes are seeking for shared resources at the same time. There is also the possibility of competition for shared global resources between various apps running on various nodes.

Previous research [19, 20] focuses on resolving the issue of straggler tasks by determining and minimizing which activities are stragglers only after all activities have been completed. This approach was taken to solve the problem of straggler tasks. The term "straggler mitigation" refers to the process of preventing any influence that straggler tasks may have on quality of service. This not only needs constant computation resources but also these tracking tasks themselves may be so data-intensive that it may lead to resource conflict, slowdowns, and preclude the throughput of the system [26]. This not only needs constant computation resources but also these monitor and control activities themselves can be so data-intensive. However, contemporary technologies such as deep learning make it possible to construct scalable models that can not only detect but also anticipate in advance which jobs might be lagging behind, enabling the execution of mitigation methods that save time and enhance quality of service. In this context, "straggler prediction" refers to the process of forecasting straggler tasks before they are carried out. In particular [27, 23] deploy solutions that utilize deep learning to anticipate lagging chores and handle them in an effective manner.

Methods of straggler prediction that are based on deep learning are susceptible to significant prediction errors for two primary reasons. To begin, these models do not take into account the underlying distribution of the periods it takes to complete activities, which is an essential factor in identifying lagging tasks [20, 29]. In particular, the existence of tasks with exceptionally high or low running time is caused by the fact that there is variability in the times at which jobs are completed. When simulating the dispersion of task turnaround time, this results in a relatively wide state space for the neural network, and as a result, it is frequently left out of practical approaches [27]. Furthermore, these methods do not take into account the different capacities of the hosts, which can also result in poor sequencing or decisions on prevention [26]. As a result, a new strategy is required, one that is capable of both proactively predicting straggler tasks and effectively mitigating their effects. Fog-cloud environments are one type of diverse runtime environment [26]. These environments take advantage of the resource capabilities offered by edge devices in addition to cloud nodes. Because of this, the computing resources of different host devices within the same context are very different from one another. This host heterogeneity has a consequence on the response time, as the scheduling process in a device with limited resources might dramatically lengthen the device's reaction time.

Because of these problems, it is necessary to design an innovative method of hybrid machine learning in order to identify and eliminate the straggler nodes. The system that has been proposed is able to perform an assessment on the current condition of a cloud environment. Utilization of resources like CPU, RAM, disc space, and bandwidth are some of the host and task characteristics that are used to characterise the current state of the cloud system. Previous research [30] served as inspiration for these parameters. In addition, previous research [20] has demonstrated that the reaction times of jobs in large-scale cloud installations follow a Pareto distribution. The suggested hybrid machine technique is utilised to forecast this distribution in advance in order to eliminate the straggler problem in a proactive manner. During the running of jobs, the speculative and rerun-based approaches that are utilised by the proposed HML are utilised for the purpose of Straggler Mitigation. Early mitigation is possible because to prediction, which also helps cut down on execution time while keeping quality of service at the appropriate depth. A comparison is made between the proposed method and certain well-known existing methods (SVM, ADABOOST, ANN, NB, DT, and RF) in terms of quality-of-service factors such as energy usage, processing time, resource contention, and CPU use. The results of the experiments show that the suggested use of HML results in a shorter amount of time needed for the execution than the currently used methods, while also providing a minimal amount of computational overhead. The remaining parts of the paper are organised as described below. The related work is presented in Section 2. Section 3 details framework and algorithm of proposed hybrid machine technique. The assessment setting as well as the experimental outcomes are discussed in Sections 4. The discussion is brought to a close in Section 5, which also provides an overview of potential future research topics.

RELATED WORK

A unique NIDS framework utilizing a deep convolution neural network that makes use of network spectrogram images obtained using the short-time Fourier transform is proposed by Adnan Shahid Khan et al. [1]. The CIC-IDS2017 dataset served as the basis for evaluation of the effectiveness of the approach that is been proposed. When compared to previous Deep Learning (DL) methods, the experimental findings showed an improvement of approximately 2.5%–4% in properly detecting intrusions. At the same time, the FAR was reduced by 4.3%–6.7% when considering a binary classification situation. Its effectiveness is also noticed for a 7-class categorization scenario, where it achieved about 98.75% accuracy with an improvement of 0.56% 3.72% in comparison to previous DL approaches.

Sana Ullah Jan et al. [2] suggest a distributed sensor-fault detection approach in the year 2020. The system is built on machine learning methods, and the fault detection block is integrated in the sensor so that output can be achieved instantly after data collection. The effectiveness of fault detection is evaluated using a number of different metrics, including detection accuracy, the area beneath the ROC curve (AUC-ROC), the percentage of false positives, and the F1 score. In addition to this, the classifier performance parameter demonstrates how effective the defect identification process is. The outcomes of the experiments demonstrate that the suggested fuzzy learning based model is superior to traditional neuro-fuzzy & non-fuzzy learning methods in terms of efficiency.

Fault detection via a wireless sensor network in a completely decentralized way is the topic of a paper that was published in 2021 by R. Regin et al. [3]. Firstly, the Convex Hull method is suggested to compute a collection of extreme ends with the neighboring nodes, and the length of the message is constrained to remain the same even as the amount of nodes rises. Second, it is recommended to use a Naive Bayes classifier in conjunction with a CNN in order to increase the convergence performance and locate the node flaws. In the end, the convex hull, Naive bayes,

and CNN techniques are evaluated with real-world datasets in order to discover and organize the flaws. Both results from simulation and experiments confirm the technique's practicality and efficiency, and demonstrate that, based on performance measures, the CNN approach contains defects that are more easily discovered than those in the convex hull approach.

An efficient fault detection, energy-efficient, quality-of-service routing strategy dependent on reinforcement learning is presented by Tariq Mahmood et al. [4] in the year 2021. The goal of this technique is to find the optimal route with the lowest number of end-to-end delay possible. Furthermore, the selection of the cluster head is contingent on the residual energy that is produced by the cluster nodes, which in turn reduces the existence of the entire network. As a consequence of this, it lengthens the lifespan of the network, reduces the amount of energy that is consumed during data transmission, and increases the resiliency of the network. The findings of the experiments reveal that the effectiveness of the network has been successfully improved by fault-tolerance solutions that incorporate trustable computational resources, which has resulted in a decreased risk of network problems.

In 2022, S. Gnanavel et al. [5] Classification techniques are utilized in a WSN to identify errors for the purpose of quality assurance checks on the data produced by the sensor network. For the purpose of this study, six different classifiers, including SVM, CNN, Multilayer Perceptron, Stochastic Gradient Descent (SGD), Random Forest (RF) and Probabilistic Neural Network (PNN) were used. The information that is produced by the sensor nodes has a variety of errors introduced into it, including an Offset fault, a Gain fault, a Stuck-at fault, an Out of Bounds fault, a Spike fault, and a Data loss fault. Classification methods do quality assurance checks on the inaccurate data. The results of the simulation demonstrate that the RF discovered more errors than any other classification in that class, and it also performed better than any of those other classifiers.

In 2020, Mohammad Reza Samsami et al. [6] provide a survey of the role of the distributed approaches in DRL. It overviews the state of the field, by studying the key research works that have a significant impact on how we can use distributed methods in DRL. The overview of papers were chosen, from the perspective of distributed learning, and not the aspect of innovations in reinforcement learning algorithms. Also, these methods were evaluated on different tasks, and compare their performance with each other and with single actor and learner agents.

In 2020, Kun Yang et al. [7] Although many distributed denial of service (DDoS) attacks detection algorithms have been proposed and even some of them have claimed high detection accuracy, DDoS attacks are still a major problem for network security. The latent and inherent problems of these detection algorithms are 1) Requirement of both normal and attack data for building detection models, and 2) Almost inability to detect novel and unknown DDoS attacks. To conquer the problems, this paper proposes an AutoEncoder based DDoS attacks Detection Framework (AE-D3F), which only uses normal traffic to build the detection model and is able to update itself automatically as time goes. Experimental results on synthetic and public traffic show that our AE-D3F can not only achieve 82.00% detection rate (DR) with 0 false positive rate (FPR), better than classical anomaly detection approaches, but also detect novel and unknown attacks.

In 2016, Jiabin Li et al. [8] propose a detection method that consists of 3 main parts in different aspects: a sliding time window to fasten the entropy calculation, a single-directional filter to realize early detection during the DDoS progress but not after the crash, and a quintile deviation check algorithm to optimize the detection result. These will eventually lead to a real-time and high-efficient performance to recognize IoT DDoS attacks as soon as possible.

In 2020, Rabindra Kumar Shial et al. [9] proposed a centralized faulty node detection algorithm based on statistical analysis in wireless sensor network. The proposed algorithm is evaluated and simulation result shows that the algorithm performs better than the existing conventional approaches.

In 2020, Shrishti Sajan et al. [10] Wireless Sensor Networks (WSNs) is a fundamental apparatus for monitoring discrete remote situations. One of the key innovations engaged with WSNs, is fault recognition in WSN applications. WSN are usually fault prone and1reliability of1sensor network1is influenced1by flaws that might1occur, because of different reasons like malfunctioning hardware and also software faults or due to some natural reasons. The primary motive of this paper is to consider various approaches to fault detection techniques in WSNs and their upcoming predictions. To achieve this point, various existing approaches are reviewed and provided a broader outline of fault detection and also fault tolerance in WSNs. In this paper, the summarization of the existing fault detection techniques is stated, and further examinations are done to help sensor applications.

In 2020, Anwesha Das et al. [11] presented the framework Aarohi, which describes an effective way to predict failures online. Aarohi is designed to be generic and scalable making it suitable as a real-time predictor. Aarohi obtains more

than 3 minutes lead times to node failures with an average of 0.31 msecs prediction time for a chain length of 18. The overall improvement obtained w.r.t. the existing state of-the-art is over a factor of 27.4×. The compiler-based approach provides new research directions for lead time optimization with a significant prediction speedup required for the deployment of proactive fault tolerant solutions in practice.

In 2021, Jiaxi Liu et al. [12] Many environmental monitoring applications that are based on the Internet of Things (IoT) require robust and available systems. These systems must be able to tolerate the hardware or software failure of nodes and communication failure between nodes. However, node failure is inevitable due to environmental and human factors, and battery depletion in particular is a major contributor to node failure. The existing failure detection algorithms seldom consider the problem of node battery consumption. In order to rectify this, a low-power failure detector (LP-FD) is proposed that can provide an acceptable failure detection service and can save on the battery consumption of nodes. From simulation experiments, results show that the LP-FD can provide better detection speed, accuracy, overhead and battery consumption than other failure detection algorithms.

In 2020, Naoto Numata et al. [13] proposes an IP fast reroute method which can reroute packets against multiple node failures. The paper is the first paper which deals with multiple node failures in the research area on IP fast reroute. The proposed method generates spanning trees to bypass the failures from a given network topology in network design stage, and reroutes a packet using one of the generated spanning trees every time the packet encounters a node failure in network operation stage. Numerical example shows that such spanning trees can be easily generated using our proposed method.

In 2020, Mridula Dhingra et al. [14] the growth of cloud computing is rapid and consumers expect additional resources and improved results, so the load balance of cloud computing has come to be very well known. Load balancing is essential in the distributed environment for an efficient operation. It helps achieve a high level of customer loyalty and utilization of resources by certifying that all computer resources are efficiently and unbiasedly distributed. This paper discusses several algorithms to present competent methods for increasing Cloud presentation in its entirety, providing the customer with a more appropriate and competent environment.

In 2020, Liangliang Xu et al. [15] propose PDL, a PBD-based Data Layout, to optimize failure recovery performance in DSSes. PDL is constructed based on Pairwise Balanced Design, a combinatorial design scheme with uniform mathematical properties, and thus presents a uniform data layout. Then it propose rPDL, a failure recovery scheme based on PDL. rPDL reduces cross-rack traffic effectively and provides nearly balanced cross-rack traffic distribution by uniformly choosing replacement nodes and retrieving determined available blocks to recover the lost blocks. The PDL and rPDL is implemented in Hadoop 3.1.1. Compared with existing data layout of HDFS, experimental results show that rPDL reduces degraded read latency by an average of 62.83%, delivers 6.27× data recovery throughput, and provides evidently better support for front-end applications.

In 2021, Chitturi Sai Nikhil et al. [16] Identification of Node failure detection and a localization is a very important challenge in a network community to get a quick recovery and avoid useless traffic in network. But it is very difficult to check the failure nodes or locations because of the large number of Screw ups in dense network. As finding the main source for failure of network is always challenging the proposed work will achieve that, it identifies the node failure by using probing measurement of binary state to end to end paths. Apart from identifying the network failure, it also quantifies the total failure nodes and the ip address or vicinity of failure nodes, Identification of node failure is done by monitoring nodes which are deployed in the network. The Proposed word is divided majorly in two phases one is identifying the node failures by using Probing Packets and other is finding of the failure and its recovery.

In 2020, S. Siva Rama Krishnan et al. [17] Wireless sensor networks are used to monitor physical or environmental conditions such as temperature and pressure as well as to study the quality of certain environmental and natural entities like air and water bodies by collecting data about the various components present in the air/water at a given spot and time. But the complete data generated by the nodes in each iteration is not always useful, as most of them give the redundant information or details which does not provide any essential information, just bulge up the amount of data being transmitted. Therefore, this paper aims to formulate an early prevention method (EPM) which not only gives a way to detect failed nodes, but also increases the overall efficiency of the network by reducing the overhead at the sink.

In 2020, Baturalp Buyukates et al. [18] consider a status update system in which the update packets need to be processed to extract the embedded useful information. The source node sends the acquired information to a computation unit (CU) which consists of a master node and n worker nodes. The master node distributes the received

computation task to the worker nodes. Upon computation, the master node aggregates the results and sends them back to the source node to keep it updated. It reviewd the age performance of uncoded and coded (repetition coded, MDS coded, and multi-message MDS (MM-MDS) coded) schemes in the presence of stragglers under i.i.d. exponential transmission delays and i.i.d. shifted exponential computation times. It shows that asymptotically MM-MDS coded scheme outperforms the other schemes. Finally, the age-optimal codes are characterized.

In 2021, Aswathy Ravikumar et al. [19] Deep learning for image analytics is widely used in many real-world applications. Due to the rapid growth in data and model size there is a need to distribute the models in multiple nodes. Distributed computing of the model helps to increase the scalability, training time and its cost effectiveness. But the distribution can lead to longer computation times in case of stale nodes. The computational time of the distributed nodes are affected by many factors like latency caused due to communication, network connectivity, resource sharing, computational power etc. The main problem faced in case of distribution is the staleness among the worker nodes. Effect of stragglers cannot be completely avoided in distributed clusters. The failures in storage, disks, imbalanced workloads, resources sharing etc. are the main cause of stragglers. Stragglers can cause longer computation time and reduce the performance of the model. The different methods used to address this issue is described in the paper in detail. The open research problems in this field are also highlighted.

PROPOSED SYSTEM

As machine learning (ML) techniques grow more widespread, the process of training models will become increasingly challenging. Therefore, a distributed machine learning architecture that is easy to use, flexible, and resistant to stragglers is required. The methodology that has been proposed aims to reduce the consequences that are caused by stragglers in huge training activities. In synchronous distributed computing, lagging workers are referred to as stragglers. Stragglers fall behind the rest of the workers. In this section, the methodology that is proposed for minimising the consequences of stragglers in decentralized machine learning is outlined. Straggler nodes are the cause of delays in synchronous processing. This takes place before proceeding onto another stage of the computation, and it requires the findings of all of the workers to be integrated.

The proposed HML for straggler detection and mitigation in highly distributed environments is illustrated in Figure 1. The data has been evaluated in accordance with the standards and guidelines that were established during the preprocessing stage. When any of these variables exceeds or breaches the limits, the system immediately terminates the instance since it has violated the limits that have been set for it by the property's upper and lower bounds for specified values. The phases of collecting data, organizing the data, filtering and normalising it are all included in the preprocessing phase. When cleaning and repairing inaccurate or false information from files, records, or datasets, it is necessary to locate and change (or remove) any lost mistaken, erroneous, or incomprehensible information. Additionally, it is necessary to replace, update, or remove any filthy or confidential material. The proposed method cleans up the data in an informative manner by employing scripting tools or transaction processing. The consistent sampling approaches are utilised in order to equalise the data and filter the standardised dataset in order to omit the events that were improperly categorised. The standard and numerical values from the text data are retrieved in the feature extraction step of the process.

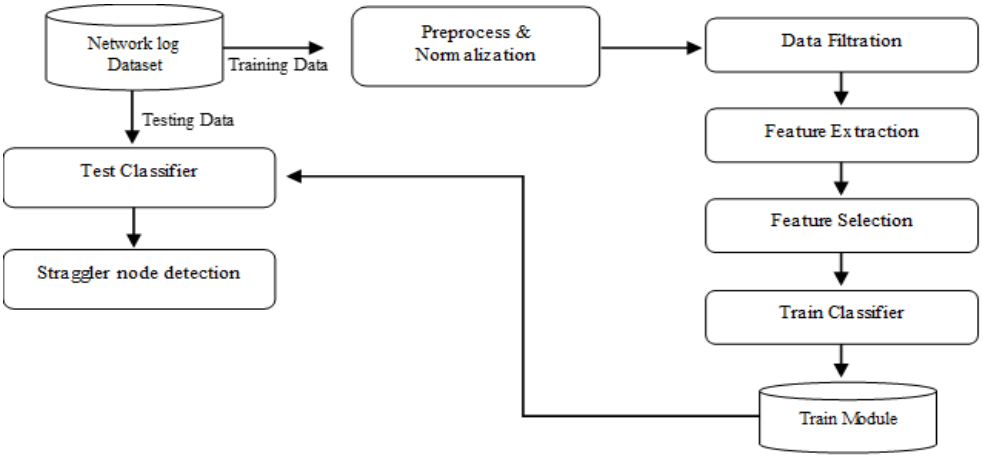


Figure 1. Proposed system architecture using HML for straggler node detection and classification

The full data set has been processed with the different feature selection and extraction strategies that have been developed. From the complete data set, the TF-IDF, N-Gram, correlation coefficients, bi-tagged and density-based features have been retrieved. After the feature extraction process has been completed, a one-of-a-kind feature vector that contains a variety of features derived from the features extracted is formed. This feature vector guarantees that it does not contain any data redundancy and offers the lowest possible level of redundancy while maintaining the highest possible level of relevance (mRmR). Using a supervised classification approach, the computer programme finds every record and determines whether it is a straggler or a normal record.

The meaning of the parameters used in the algorithm is discussed in the following table 1,

TABLE I. NOTATIONS

Symbol	Meaning
p	Max number of activities in a job
α, β	Variables of the Pareto distribution
O	Straggler variable in proposed HML
E_{St}	Anticipated number of straggler activities
I	Time-period of proposed HML inference in secs
D	Time-duration of proposed HML inference in secs
m	Total number of hosts

Algorithm of Detection and Migitation of Straggler

Inputs:

Step 1: A ← Collection of all jobs presently being performed [a_1, a_2, …, a_r]

Step 2: T_l^m ← Collection of activities of job j_m where l ∈ {1, 2, 3, 4,…p}

Step 3: M_t ← Maximum time allocated for releasing resource.

Parameters:

Step 4: A_m ← Collection of normal jobs ⊆ A without any activities of straggler

Step 5: A_s ← Collection of jobs ⊆ A with > 0 activities of straggler

Procedure ForecastStraggler (job)

Step 6: for time d from 0 to D by using step 1

Step 7: p ← Number of activities in input job

Step 8: Retrieve feature vectors of host systems as M_{Host}

Step 9: Retrieve feature vectors of activities of input job as M_{Task}

Step 10: Forecast (α, β) using the ML

Step 11: Determine E_{St} as q $(\frac{O}{\beta})^{-\alpha}$

Step 12: Execute job until p − $\lfloor Est \rfloor$ activities are done

Step 13: return unfinished activities

Step 14: Procedure Speculation(activities list)

Step 15: for activities t in activities list

Step 16: Excecute a copy of t on another node

Step 17: Procedure RerunStragglerActivities (activities list)

Step 18: for activities t in activities list

Step 19: Execute the same activities t on another node

Step 20: Begin

Step 21: for job a_i in A

Step 22: stragglerActivities ← ForecastStraggler(a_i)

Step 23: if stragglerActivities is empty

Step 24: add a_i to A_n

Step 25: continue

Step 26: else

Step 27: add a_i to A_s

Step 28: Wait for the particular amount of time (M_t), if a_i does not reacts then the alert will generate for further action.

Step 29: if a_i is deadline driven

Step 30: Speculation (stragglerActivities)

Step 31: else

Step 32: RerunStragglerActivities (stragglerActivities)

RESULT AND DISCUSSION

Performance measurement

Utilized here are the standard metrics for doing evaluations. Let us make the assumption that there are n hosts and q jobs in the system at the moment.

Energy Usage: The total amount of energy that has been consumed over a period of time can be calculated using the formula

$$E = E_{CPU} + E_{Disk} + E_{Memory} + E_{Network} + E_{Misc,mory} + E_{Network} + E_{Misc}, \quad \ldots(1)$$

where E_{CPU} represents the total amount of energy that has been consumed by all of the processors and comprises dynamic energy as CV ²f, short-circuit energy, discharge energy, and the amount of energy that has been consumed while the processors have been idle [10]. E_{Disk} represents the total amount of energy used by all read/write activities as well as the energy used by all discs while they are idle. E_{Memory} refers to the amount of energy that is utilised by the RAM as well as cache memory included in the computing nodes. $E_{Network}$ refers to the aggregate amount of energy that is used by network equipment such as routers, ports, LAN adapters, and switches. E_{Misc} refers "other elements," which includes things like the motherboards and port connector. To calculate the maximal and minimal energy usage (E_{max}, E_{min}), hardware profiling is used in accordance with Equation 1. After that, Equation 2 is used to calculate the total amount of energy consumed at period t. In this context, total host resource consumption, or U_k^t refers to host k's whole tasks combined. This is a typical method [27]. Thus,

$$E_{total}^t = \sum_{k=1}^{n} U_k^t \cdot (E_{max} - E_{min}) + E_{min}. \quad \ldots(2)$$

Execution Duration: The average execution duration is computed by the following formula,

$$T_{avg}^{exec} = \frac{1}{q} \sum_{i=1}^{q} (T_i^C - T_i^S) + \sum_{i=1}^{q} R_i. \quad \ldots(3)$$

This is the entire amount of time, on average, that it takes to correctly perform an application for each and every task. Here T_i^C, T_i^S and R_i represent the time at which task i was finished, submitted, and restarted, respectively.

Resource Contention: The term "resource contention" refers to the situation in which two or more tasks utilize the same resource while it is being executed [20]. This could be because the required quantity of resources are not readily available, or it could be because there is an excessive amount of work to be done with stringent due dates. Resource contention is measured as

...(4)

$$Con_{total}^{resource} = \sum_{k=1}^{n} \sum_{i=1}^{q_k} Req_{i,k}^{resource} \cdot \mathbb{1}(resource_i \; overloaded),$$

where the amount of jobs being carried out at resource k is denoted by q_k and the resource need of the i^{th} task carried out at node k is denoted by $Req_{i,k}^{\text{resource}}$. Additionally, the indicator function is denoted by the 1() notation.

Memory Usage: The memory usage of host k is computed by,

$$U_k^{memory} = \frac{P_k^{total} - (F_k + B_k + C_k)}{P_k^{total}} \times 100, \quad \text{...(5)}$$

Where P_k^{total} indicate the total amount of physical memory, F_k is the amount of free memory, B_k is the buffer size, and C_k is the cache size.

Disk Usage: The disk usage of host k is computed by

...(6)

$$U_k^{disk} = \frac{Total\,Used}{Total\,HD\,Size} \times 100.$$

Network Usage: The network usage of host k is computed by

...(7)

$$U_k^{network} = \frac{Bits_{total}^{rx} + Bits_{total}^{tx}}{BW_k \times S_I} \times 100,$$

Where the total bits collected and transferred in an interval are indicated by the variables $Bits_{total}^{rx}$ and $Bits_{total}^{tx}$ respectively. The bandwidth of host k is denoted by BW_k, and the duration of the interval is denoted by S_I.

Experimental Observations

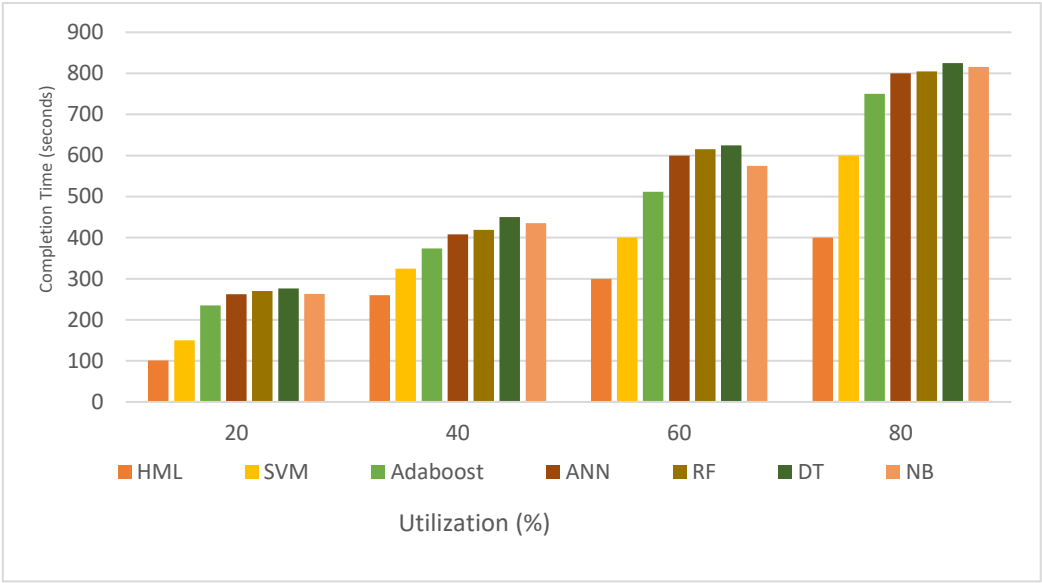
The performance of the proposed HML is evaluated in comparison to the approaches that are already in use with the help of the QoS metrics. The studies are carried out over the course of a full day, which corresponds to 288 scheduling intervals. A total of five runs were averaged, and a variety of job kinds were employed to guarantee that the results were statistically significant.

Utilization of Resources on a Variable Scale

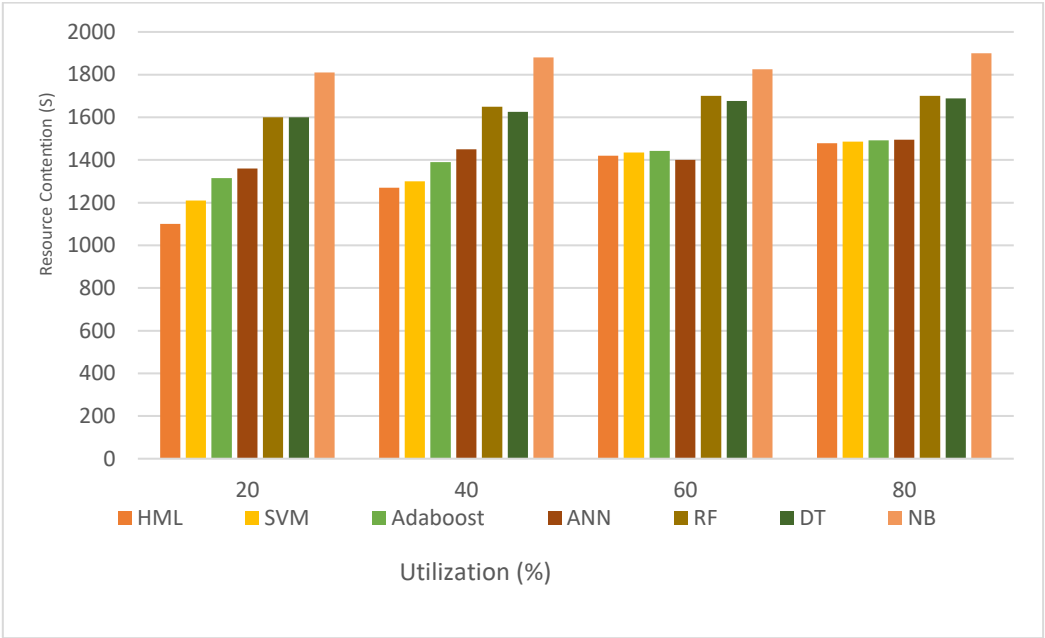
For the purpose of evaluating how well the suggested method performs, we took into account four distinct types of reserved usage for the CPU, the disc, the memory, and the network. These types involve blocking use on purpose at 20%, 40%, 60%, and 80% respectively. Figure 6 depicts a comparison of various QoS characteristics, including Completion Time, Energy usage and Resource Contention, with varying values of Cpu Usage, Disk Utilization, Network Utilization, and Memory Consumption.

The values of completion time for various straggler management techniques are depicted in Figure 2(a), along with variations in the values of the percentages of CPU, disc, network, and memory use. The value of runtime rises along with the value of reserved utilisation, but the performance of the proposed HML is superior to that of the techniques that are currently in use because it monitors the states of the resources in a dynamic manner in order to make decisions that are more effective. In comparison to the baseline approaches, the measure of completion time in the suggested HML takes 11.47-17.4% lesser time. The fluctuation in the amount of competition for a resource is depicted in Figure 2(b), which shows how utilisation can take on a variety of values. When there is greater demand for a

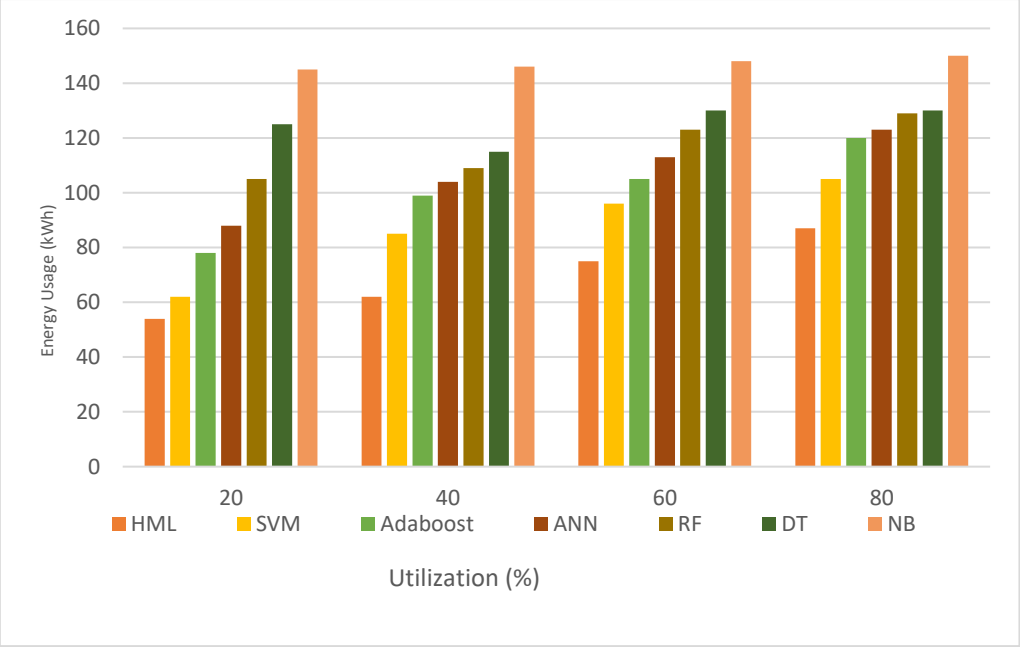
resource, there is a corresponding rise in the value of resource dispute. When compared to the baseline techniques, the value of resource conflict in the proposed HML is between 12.34 and 15.19% lower. Figure 2(c) illustrates the energy usage for various values of utilisation, and the observations reveal that the energy usage rises in line with the level of utilisation across the board for straggler management strategies. However, in comparison to the state of the art, the proposed HML operates significantly better because it prevents the over- or below of resources while scheduling. When compared to the baseline approaches, the value of energy usage in the proposed HML is lower by between 18.55% and 22.43%.



(a)



(b)



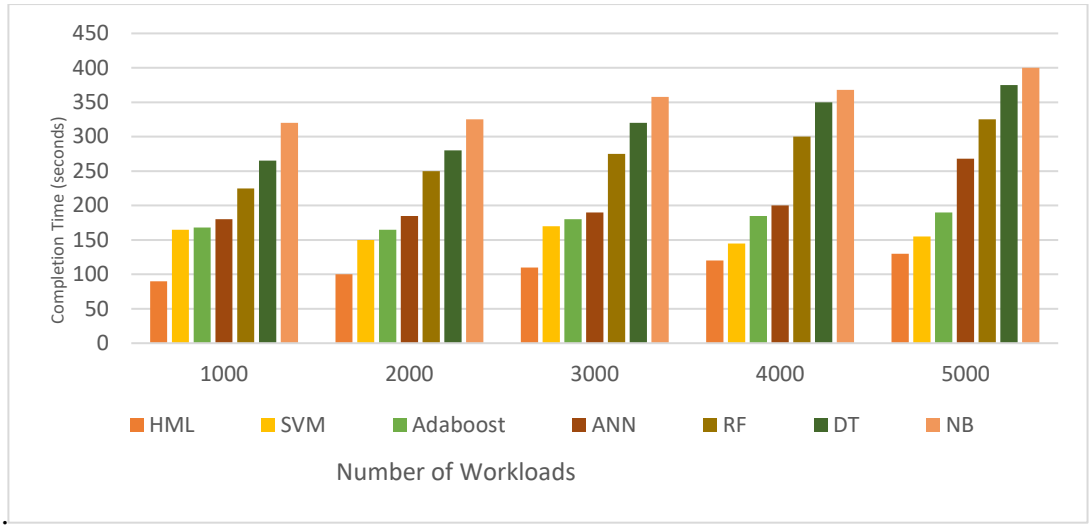
(c)

Figure 2. Comparing QoS parameters with various value of CPU usage, disk usage, network usage and memory usage: a) Completion Time, b) Resource Contention, c) Energy Usage

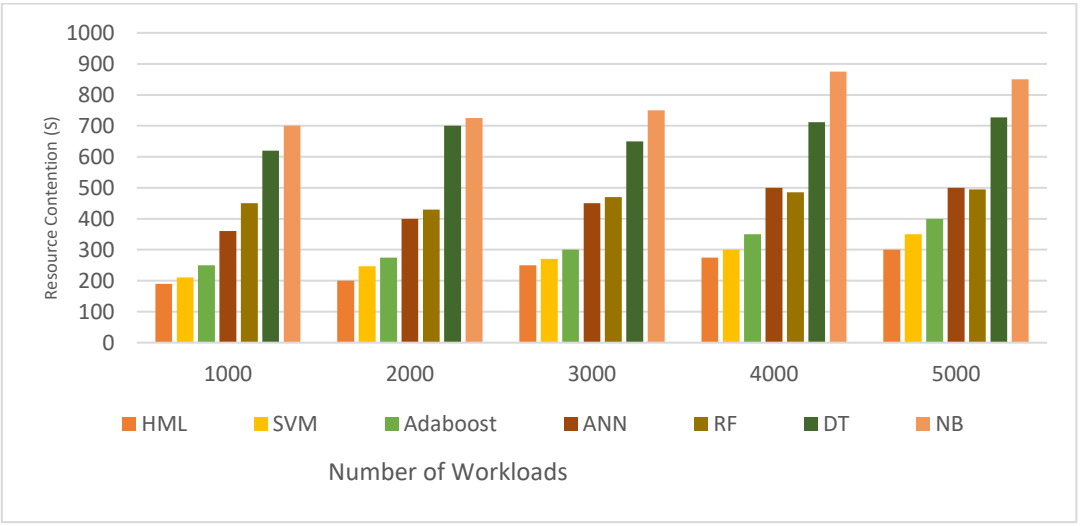
Variation of Number of Workloads

In this section, the significance of various indicators of performance is evaluated as the amount of workloads increases. This assessment is made in relation to the previous section.

The difference in execution time is seen in Figure 3(a), which corresponds to varied amounts of workloads. In comparison to the baseline approaches, the value of processing time in the suggested HML takes 19.74-23.84% less time. The value of resource conflict rises as the amount of workloads rises, as illustrated in Figure 3(b), which provides an assessment of resource conflict for various numbers of workloads. This analysis reveals that resource conflict is a function of the number of workloads. The performance of the proposed HML is superior to that of the current methods; the mean value of resource conflict in the proposed HML is 19.12-24.84 percentage points lower than that of the conventional classifiers. Figure 3(c) illustrates how the value of energy usage can vary depending on the number of workloads, and the value of energy usage in the proposed HML is 13.71-18.01% lower than the value of energy usage in the baseline techniques. The difference in network utilisation with a varying number of workloads is displayed in Figure 3(d), which compares the proposed HML approach to the conventional classifiers. Every utilisation measure that is displayed in the chart is an average across all of the jobs that have been finished. The findings of the experiments indicate that the suggested HML has an overall average of network utilisation that is around 18.6% and 25.67% higher than the approaches that were used as a baseline. Figure 3(e) depicts the fluctuation of CPU utilisation with various numbers of workloads. It demonstrates that the value of CPU utilisation is declining with the rise in the number of tasks, but the suggested HML outperforms than other strategies that are already in use. When compared to the approaches used as a baseline, the value of CPU consumption in the proposed HML is somewhere between 16.61% and 17.29% higher. The difference in disc use across all techniques is depicted in Figure 3(f), which indicates the effect that changing the amount of workloads has. The research results show that the suggested HML has an overall average of disc consumption that is 13.25-15.34% higher than the approaches that serve as the baseline. The value of memory utilisation is reducing with the increase in the amount of tasks, as illustrated in Figure 3(g), but the proposed HML accomplishes superior to existing techniques. This is indicated by the fact that the variability of memory utilisation with a varying amount of workloads is depicted in figure. Memory use in the proposed HML is 7.92-17.54% higher than the value of memory utilisation in the baseline approaches. In the case of the proposed HML, the more conservative completion of tasks that is based on straggler forecasting is the reason for the reduction in the amount of resources that are used. In order to prevent wasting resources and ensure that the forecasted straggler jobs are completed on time, those tasks are not duplicated if they are finished earlier than planned.


450
400
350
300
250
200
150
100
50
0
Completion Time (seconds)
1000
2000
3000
4000
5000
HML
SVM
Adaboost
ANN
RF
DT
NB
Number of Workloads

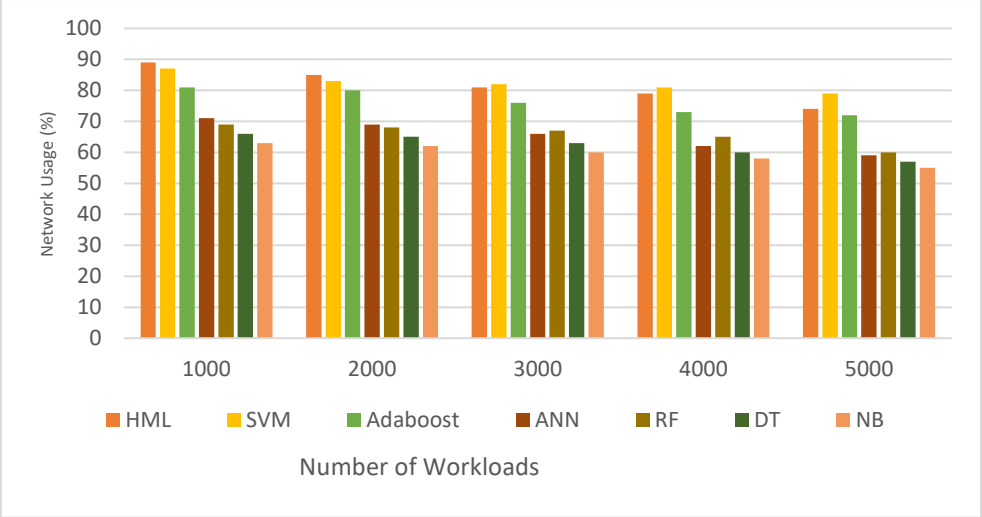
(a)


1000
900
800
700
600
500
400
300
200
100
0
Resource Contention (S)
1000
2000
3000
4000
5000
HML
SVM
Adaboost
ANN
RF
DT
NB
Number of Workloads

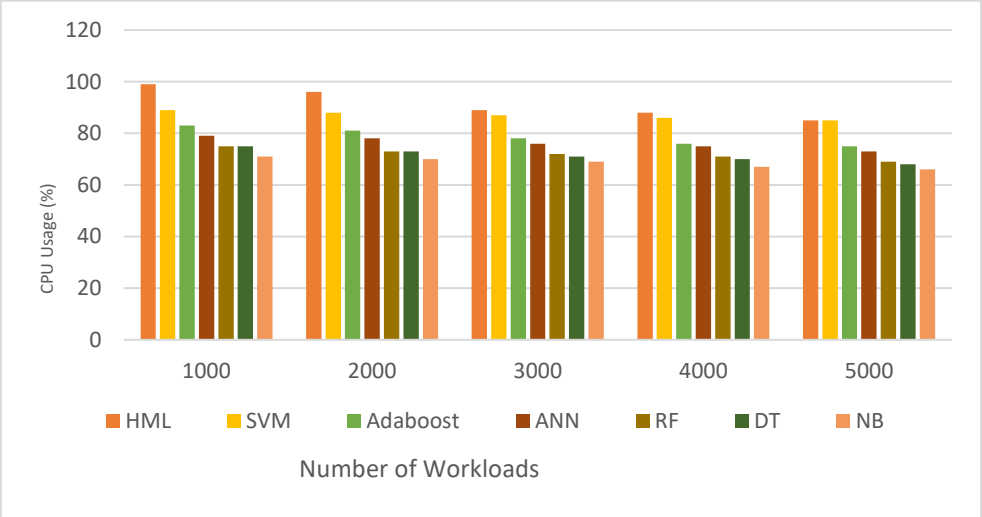
(b)


250
200
150
100
50
0
Energy Usage (kWh)
1000
2000
3000
4000
5000
HML
SVM
Adaboost
ANN
RF
DT
NB
Number of Workloads

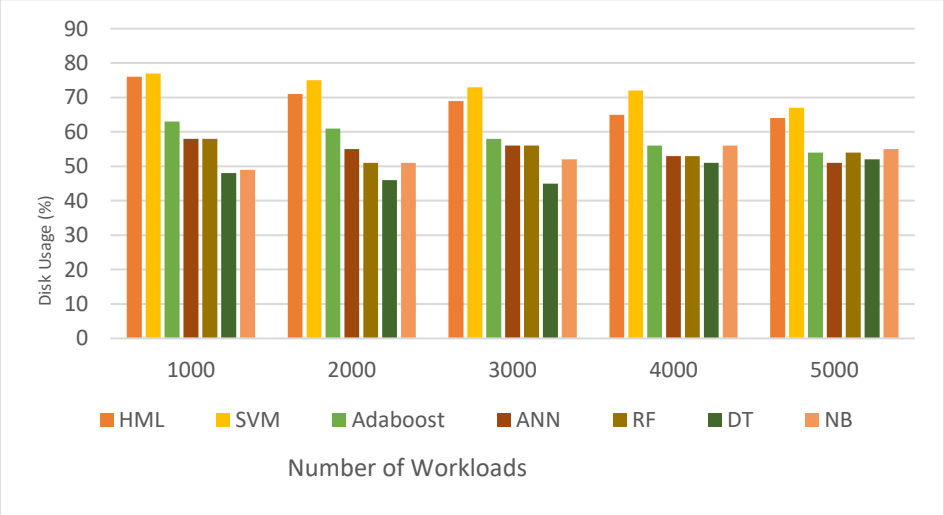
(c)


100
90
80
70
60
50
40
30
20
10
0
Network Usage (%)
1000
2000
3000
4000
5000
HML
SVM
Adaboost
ANN
RF
DT
NB
Number of Workloads

(d)


120
100
80
60
40
20
0
CPU Usage (%)
1000
2000
3000
4000
5000
HML
SVM
Adaboost
ANN
RF
DT
NB
Number of Workloads

(e)


90
80
70
60
50
40
30
20
10
0
Disk Usage (%)
1000
2000
3000
4000
5000
HML
SVM
Adaboost
ANN
RF
DT
NB
Number of Workloads

(f)

(g)

Figure 3. Comparing parameters of performance with various value of workloads: a) Completion Time, b) Resource Contention, c) Energy Usage, d) Network Usage, e) CPU Usage, f) Disk Usage and g) Memory Usage

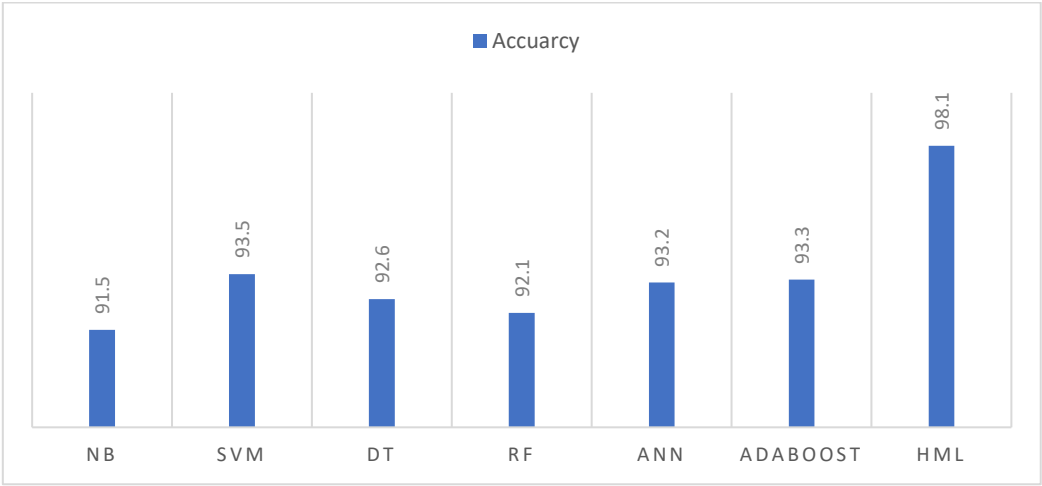


Figure 4. Accuracy of proposed HML compared with conventional ML Classifiers

The accuracy of proposed HML is compared with the various conventional machine learning classifiers in figure 4 and found that proposed HML outperforms than other existing ML approaches.

CONCLUSION AND FUTURE SCOPE

It is believed that large-scale cloud computing settings can benefit from a unique straggler detection and mitigation strategy that uses hybrid HML. This method can shorten the amount of time needed to respond while also producing superior results to those of earlier efforts. The approach that has been suggested is able to accurately predict straggler jobs in advance and eliminate those utilising methods such as prediction and re-running at an earlier stage. The proposed HML, in contrast to earlier prediction-based strategies, is able to analyse tasks in conjunction with host features and make use of the fundamental Pareto distribution in order to make more accurate predictions and take preventative measures, which ultimately results in higher performance than current state-of-the-art mechanisms. It is abundantly obvious that the suggested performs better across a variety of workload levels, resulting in reduced completion time, resource contentions, and energy usage. The performance of the proposed method again outperforms that of the baseline approaches when evaluated with various levels of workload on the cloud system. The proposed HML has a higher utilisation of the CPU, network, RAM, and disc. This is due to the fact that numerous jobs, and consequently tasks, are completed in a short amount of time, which leads to a greater number of tasks being completed in a specified period of time compared to certain other ways. Even with somewhat higher resource consumption for the same amount of jobs, this demonstrates that the proposed HML is able to utilise resources in a more effective manner, resulting to faster completed work and, as a result, also conserving energy. When compared to other traditional ML classifiers, the suggested HML has a performance accuracy of 98.1%, making it superior to those other

methods. The proposed HML can be applied in real-world contexts by making use of fog frameworks as part of ongoing research and development. This will contribute to making the model highly resistant to the stochasticity of tasks and workloads that occur in real-world situations. In addition, the suggested HML has the capability of being fine-tuned by making use of a bigger dataset That contains a Variety of Cloud and Fog Applications.

REFERENCES

[1] Khan, A.S., Ahmad, Z., Abdullah, J., Ahmad, F. (2021). A spectrogram image-based network anomaly detection system using deep convolutional neural network. IEEE Access, 9: 87079-87093. https://doi.org/10.1109/ACCESS.2021.3088149.

[2] Jan, S.U., Lee, Y.D., Koo, I.S. (2021). A distributed sensor-fault detection and diagnosis framework using machine learning. Information Sciences, 547: 777-796. https://doi.org/10.1016/j.ins.2020.08.068.

[3] Regin, R., Rajest, S.S., Singh, B. (2021). Fault detection in wireless sensor network based on deep learning algorithms. EAI Endorsed Transactions on Scalable Information Systems, 8(32): e8-e8. https://doi.org/10.4108/eai.3-5-2021.169578.

[4] Mahmood, T., Li, J., Pei, Y., Akhtar, F., Butt, S.A., Ditta, A., Qureshi, S. (2022). An intelligent fault detection approach based on reinforcement learning system in wireless sensor network. The Journal of Supercomputing, 78(3): 3646-3675. https://doi.org/10.1007/s11227-021- 04001-1.

[5] Gnanavel, S., Sreekrishna, M., Mani, V., et al. (2022). Analysis of fault classifiers to detect the faults and node failures in a wireless sensor network. Electronics, 11(10): 1609. https://doi.org/10.3390/electronics11101609.

[6] Samsami, M.R., Alimadad, H. (2020). Distributed deep reinforcement learning: An overview. arXiv preprint arXiv:2011.11012. https://arxiv.org/abs/2011.11012.

[7] Yang, K., Zhang, J., Xu, Y., Chao, J. (2020). DDoS attacks detection with autoencoder. In NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, pp. 1-9. https://doi.org/10.1109/NOMS47738.2020.9110372.

[8] Li, J., Liu, M., Xue, Z., Fan, X., He, X. (2020). RTVD: A real-time volumetric detection scheme for DDoS in the Internet of Things. IEEE Access, 8: 36191-36201. https://doi.org/10.1109/ACCESS.2020.2974293.

[9] Shial, Rabindra Kumar, et al. "A Centralized Faulty Node Detection Algorithm Based on Statistical Analysis in WSN." 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA). IEEE, 2020.

[10] Sajan, Shrishti, et al. "Performance Evaluation Of Various Algorithms That Affect Fault Detection In Wireless Sensor Network." 2020 Fourth International Conference on Inventive Systems and Control (ICISC). IEEE, 2020.

[11] Das, Anwesha, Frank Mueller, and Barry Rountree. "Aarohi: Making Real-Time Node Failure Prediction Feasible." 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2020.

[12] Liu, Jiaxi, et al. "Low-Power Failure Detection for Environmental Monitoring Based on IoT." Sensors 21.19 (2021): 6489.

[13] Numata, Naoto, et al. "An IP Fast Reroute Method against Multiple Node Failures." 2020 International Conference on Information and Communication Technology Convergence (ICTC). IEEE, 2020.

[14] Dhingra, Mridula, and Neha Gupta. "Failure Node Reduction Algorithm to Enhance Fault Tolerance Capability of Cloud Nodes." 2020 12th International Conference on Computational Intelligence and Communication Networks (CICN). IEEE, 2020.

[15] Xu, Liangliang, et al. "PDL: A Data Layout towards Fast Failure Recovery for Erasure-coded Distributed Storage Systems." IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE, 2020.

[16] Nikhil, Chitturi Sai, et al. "Efficient identification of node failure and recovery through end to end Probing techniques." Journal of Physics: Conference Series. Vol. 2040. No. 1. IOP Publishing, 2021.

[17] Krishnan, S. Siva Rama, and Arunkumar Thangavelu. "An early prevention method for node failure in wireless sensor networks." International Journal of Internet Technology and Secured Transactions 10.5 (2020): 507-537.

[18] Baturalp Buyukates, Sennur Ulukus," Timely Updates in Distributed Computation Systems with Stragglers", 2020 54th Asilomar Conference on Signals, Systems, and Computers, IEEE.

[19] Aswathy Ravikumar and Harini Sriraman, "Staleness and Stagglers in Distibuted Deep Image Analytics", International Conference on Artificial Intelligence and Smart Systems (ICAIS-2021) IEEE.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] S. S. Gill, S. Tuli, M. Xu, I. Singh, K. V. Singh, D. Lindsay, S. Tuli, D. Smirnova, M. Singh, U. Jain et al., "Transformative effects of iot, blockchain and artificial intelligence on cloud computing: Evolution, vision, trends and open challenges," Internet of Things, p. 100118, 2019.

[27] Y. Lu, L. Liu, J. Panneerselvam, B. Yuan, J. Gu, and N. Antonopoulos, "A gru-based prediction framework for intelligent resource management at cloud data centres in the age of 5g," IEEE Transactions on Cognitive Communications and Networking, 2019.

[28] S. Tuli, N. Basumatary, S. S. Gill, M. Kahani, R. C. Arya, G. S. Wander, and R. Buyya, "Healthfog: An ensemble deep learning based smart healthcare system for automatic diagnosis of heart diseases in integrated iot and fog computing environments," Future Gerenation Computer Systems, vol. 104, pp. 187–200, 2020.

[29] S. S. Gill, S. Tuli, A. N. Toosi, F. Cuadrado, P. Garraghan, R. Bahsoon, H. Lutfiyya, R. Sakellariou, O. Rana, S. Dustdar et al., "Thermosim: Deep learning based framework for modeling and simulation of thermal-aware resource management for cloud computing environments," Journal of Systems and Software, p. 110596, 2020.

[30] S. Tuli, S. Poojara, S. N. Srirama, G. Casale, and N. Jennings, "COSCO: Container Orchestration using Co-Simulation and Gradient Based Optimization for Fog Computing Environments," IEEE Transactions on Parallel and Distributed Systems, 2021.