Volume 70, Issue 3, 2026
Journal of Scientific Research
of
The Banaras Hindu University

Failure Forecasting and Adaptive Mitigation Smart Scheduling Techniques in Cloud

Mohit Gokhroo[1], Avinash Panwar[2]

[*1]Department of Computer Science MLSU Udaipur India, gokhroo@mlsu.ac.in
[2]Department of Computer Science MLSU Udaipur India, avinash@mlsu.ac.in

Abstract: **Modern day datacenters that power cloud infrastructure experience performance degradation due to unstable resource utilization patterns resulting in failing jobs. This is due to lack of dynamism in scheduling techniques. Traditional failure prediction and scheduling approaches mainly rely on static CPU and Memory usage metrics based binary prediction models, which fail to capture dynamic fluctuations in workload behavior. The dynamic features like CPU and memory volatility plays a significant role in identifying unstable machine behavior prior to failure occurrence. This work proposes Failure Forecasting and Adaptive Mitigation Smart Scheduling Techniques (FFAMSST) for Cloud Environment. This proactive cloud reliability framework integrates machine learning ensemble models that uses the learning of multiple machine learning models for the purpose of forecasting failures is to generate accurate failure risk scores, lead-time estimations and criticality-aware adaptive migration scheduling. Failure risk is used to take adaptive mitigation actions including strict monitoring, speculative replication, and live task or machines migration to next available good machines. The proposed scheduler dynamically adapts by implementing a mitigation strategy according to workload criticality, estimated lead time, and CPU volatility features from the input. The proposed framework FFAMSST leverages metrics as CPU and memory Volatility Index (CVI and MVI) calculated from CPU and memory usage distribution and tail CPU usage distribution available in the Borg cloud dataset. The CVI and MVI quantifies temporal instability and burst behavior in systems utilization, enabling early identification of anomalous execution conditions that precede system failures. This framework is evaluated on cloud trace dataset containing 405,894 samples and 34 operational features. The results indicate that FFAMSST forecasting significantly enhances resource utilization resulting into higher task completions. Framework reduces unnecessary replication and migration overheads and enhances proactive mitigation capability. FFAMSST achieves accuracy of 99.60%, and recall of 99.20% for failure instances indicating ability to detect the majority of critical events. The false alarms are effectively controlled by precision of 98.46%. There is excellent class separability as reflected by ROC-AUC values 99.6%. PR-AUC value 99.93% confirms robustness under class imbalance. The framework significantly improves failure forecasting without introducing bias toward the majority class.**

Index Terms: **Cloud Computing, failure forecasting, Machine learning, Smart Scheduling, workload management**.

I. INTRODUCTION

Today, cloud datacenters operating from geographically distributed locations handle large-scale distributed applications like financial systems, scientific analytics, large-scale machine learning and enterprise services. Cloud workloads are highly heterogeneous and have dynamic resource requirements that fluctuates rapidly across nodes. This massive cloud scale enables fast service response to the users but at the same time it also incurs severe operational challenges associated with node instability, resource contention, workload imbalances, spikes, memory bottlenecks, and unpredictable execution failures(Jennings et al., 2015). These factors significantly affect reliability and operational cost. Even miniscule of disruptions may propagate across dependent services and significantly degrade system throughput and violate the Service Level Agreement (SLA) between service providers and users.

Reactive mitigation approaches become insufficient because corrective action is initiated only after failures occur, leading to service interruption, SLA violations, and increased recovery costs. Resource wastage becomes particularly severe when failed workloads consume resources like CPU, memory and storage for extended durations before failure detection and mitigation mechanisms manages the situations. Failure management work focuses on performance, and classification, but provides very little proactive resource management and failure mitigation mechanism in cloud environment. The scheduling algorithms are not smart enough to consider wasted resources and time. Therefore, there is a need of smart hybrid scheduling approach which can forecast failures, use CPU and memory volatility

models, derive lead time to failure, perform dynamic resource optimizations and mitigation aware scheduling for cloud infrastructures.

The motivation for proposed model is an effort using machine learning techniques to identify if a task is likely to fail during execution in a proactive manner to preserve resource wastages. It enables cloud service providers to provide higher efficient, cost effective, and reliable services delivery to the end users.

The key contributions of proposed Failure Forecasting and Adaptive Mitigation Smart Scheduling Technique (FFAMSST) designed for large-scale cluster environments are:

- The framework integrates adaptive sampling, ensemble learning, lead-time estimation, and risk-aware scheduling into a unified architecture.
- FFAMSST forecast the failure and the expected time remaining before failure occurrence. The scheduler then dynamically performs monitoring, replication or migration according to the forecasted risk level.
- It attempts to reduce unnecessary resource consumption by forecasting failures before catastrophic execution loss occurs.
- The FFAMSST use incorporates the different machine learning models and then designing an ensemble model based on the weighted approach using existing models. These all models are trained not only on static parameters but also on the dynamic parameters like CPU and MEMORY volatility features derived from the attributes in the dataset itself.
- FFAMSST uses Lead time-based failures forecasting. It is achieved by calculating the difference in the fault forecasted time by ensemble machine learning models with the actual failure time from the dataset. This lead time is then used to provide appropriate mitigation action by the smart scheduler which decides whether to simply observe the task for some more time or generate replica for task execution, do checkpointing or if the failure is inevitable then take a decision to migrate the task on to next available optimized physical machine.

The novelty in this work is the use of integrated of multiple components for a single purpose of smart scheduling which encompasses the failure forecasting by machine learning ensemble model, prediction of lead time to failure with risk aware based scheduling where Risk Score=Failure Probability + Lead Time Availability. The mitigation approach for failures is based on strategies like Monitoring, Checkpointing, Resource Scaling, Migration to next available machine. This combined approach of wholistic framework is the prime contribution.

The paper organization includes Section 2 which discusses about the existing research in the domain on forecasting, mitigation and scheduling in cloud environment. Section 3 discussed the proposed approach and methodology to solve the issues through smart scheduling. Section 4 presents experimental setup. Section 5 shows the results obtained and critical discussion about results. Section 6 concludes the work with the future scope available in this research domain

II. Literature Survey

Research on cloud failure forecasting and smart scheduling has evolved in leap and bound manner over the years. Primitive concepts of scheduling are borrowed from distributed systems which mainly focused on task execution stability and resource sharing. With the evolution of Smart Cloud Scheduling Research, the domain of cloud scheduling has evolved tremendously from its inception over the years. Scheduling objectives have changed from primitive work of straightforward allocating available resources to incoming jobs and tasks to deep analysis of available resources and incoming tasks for improving operational efficiency in under the constrained computational resources.

Initial work during early 2000s used FCFS, Round Robin, Min-Min, and Max-Min. These techniques were computationally inexpensive and practical as per the existing cloud scheduling scenarios that existed. They were effective due to their simplicity but lacked the ability to adapt dynamically. During the tenure most, scheduling decisions were static and once scheduled the decision remained unchanged throughout execution irrespective of any dynamism arising out in cloud environment. Failures were only detected and then handled accordingly. Simple recovery mechanisms were triggered if a resource became unavailable or a task terminated unexpectedly.

The idea of predicting or forecasting failures was rare. Reliability was important, but it was not yet treated as important part of scheduling objective. With the rapid expansion of cloud platforms, the cloud service providers began investing heavily in cloud research. Objectives were to solve multiple constraints and issues of operational complexity under the growing infrastructure size. Workloads became more dynamic and diverse as resource demands fluctuated more frequently.

MapReduce (Dean et al., 2004) framework improved execution reliability in large distributed clusters. Failed tasks could automatically restart on alternative machines. This reduced system interruption during workload execution. However, the framework could address recovery only after failure occurred. It did not forecast resource instability before problems actually appeared. Scheduling models (Isard et al., 2008) mainly concentrated on fairness, execution efficiency and response time (Niu et al., 2016). Reliability prediction received less attention because cloud infrastructures were still evolving. Static allocation policies renders underutilized resources and concurrent approaches used redundancy and replication (Cully et al., 2008) for hardware failures handling and workload imbalances. It led to computational overheads. The work on cloud scheduling moved from static to dynamic and adaptive resource management where allocation decisions changed according to workload demand and system conditions (Rasooli et al., 2011). Reactive mechanisms increased resource consumption during large scale execution environments. Researchers examined methods for balancing workloads across multiple computing nodes (Iyer et al., 2011). Supplementary developments in distributed systems focussed towards improving fault recovery mechanisms. Resilient distributed datasets within Apache Spark were introduced in (Zaharia et al., 2012). This approach reduced re-computation overhead during execution failure. The system improved performance for iterative processing tasks and large-scale

analytics. Even with these improvements' recovery remained reactive. Prediction models were still not integrated with scheduling decisions or mitigation planning.

Modern cloud architectures require dynamic resource management owing to highly dynamic workload demands. Around 2015 there was a constant focus on methods to estimate future behaviour of machines and jobs. Workload forecasting methods for virtualized cloud environments were discussed in (Islam et al., 2012). Static scheduling policies could not maintain both efficiency and reliability under fluctuating conditions (Singh et al., 2016). Researchers investigated predictive resource allocation to dynamically allocate the cloud resources for improving cloud efficiency according to workload behaviour (Wu et al., 2015).

The predictive scheduling and adaptive provisioning techniques used historical workload behaviour. Predictive failure analysis on Google cluster trace data was done in (Soualhia et al., 2015). Their work showed that machine learning models could identify possible task failures before execution stopped completely. This helped the scheduler take early decisions such as rescheduling tasks to other nodes. As a result, system stability improved and execution delays were reduced. Early Forecasting lead to reliable, efficient, reduced infrastructure overheads, optimized scheduling in cloud environment. Researchers worked on schemes to predict future CPU demand, memory utilization, and workload intensity. This ensured demand estimation before resource shortage actually occurred. Attempts were carried out to identify warning signals appearing prior to actual failures. Jobs and virtual machine migration were deployed as corrective mechanism when workload behaviour deviated from expected patterns. Scheduling gradually evolved from single scheduler-based resource allocation problem to a broader resource management problem at cloud service provider end.

Traditional threshold monitoring methods were unable to handle dynamic workload behaviour in large cloud systems. XGBoost (T. Chen et al., 2016) demonstrated strong capability in learning nonlinear patterns from large datasets. Their work influenced many later studies on cloud anomaly detection workload forecasting and failure prediction. Ensemble learning approaches became useful because they analysed multiple system parameters together instead of depending only on fixed threshold values. Workload forecasting and predictive resource management using machine learning approaches focused on reducing overload conditions and unnecessary resource usage. The main focus applied forecasting future workload behaviour before the performance degradation actually appeared (Yu et al., 2016; Cheng et al., 2018; Chen et al., 2023). Proactive decision making helps improve resource allocation and reduced QoS violations during high demand conditions.

It was established that proactive resource estimation reduced unnecessary allocations thus improving utilization efficiency. Researchers worked on to Energy consumption. It also became a recurring concern in the literature. Data centres were growing rapidly, and improving resource utilization was no longer viewed only as a performance issue. Virtual machine consolidation emerged from this context. Researchers carefully tried to lower energy costs without adversely affecting service quality by reducing active servers.

During the early 2000s most systems relied on static scheduling and reactive fault handling. Failures were managed after disruption occurred. Prediction mechanisms were rarely included in cloud management frameworks. Later on with the advent of adaptive scheduling system and autonomic optimization techniques can adapt to changing workload conditions for dynamic resource allocation (Ghasemi et al., 2020) to improve system efficiency and service reliability.

With the advancement of virtualization technologies resource optimization also matured and this became a innovative research direction (Li et al., 2020). Neural network models, particularly Artificial neural network (ANN) and Long Short-Term Memory (LSTM) architectures, optimization-based scheduling approaches like Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO) for multi-level resource optimization were developed with aim to balance multiple constraints that could not easily be handled by traditional heuristics methods.

After 2020 there ushered in another phase where prediction alone was not sufficient. Many frameworks combine prediction with automated decision-making (Gokhroo, 2020). A forecasting model identify a future resource bottleneck, but the scheduler is also expected to decide what action should be taken. This has led to greater interest in proactive cloud management. Works focussed on resource utilization, SLA compliance, operational cost, energy efficiency, and reliability within a single optimization framework. Several Research studies focussed on forecasting with adaptive scheduling and mitigation strategies as presented in (Rahimikhanghah et al., 2022).

The increasing complexity of cloud infrastructures encouraged researchers to explore machine learning for reliability analysis (Le Duc et al., 2019; Gollapalli et al., 2022).

Failure analysis in production data centres became another major parallel research area. Failure characteristics with strong relationships between system parameters that results into failures in cloud infrastructures were examined by (Caicedo, 2023). The major characteristics under study included CPU instability, memory pressure and scheduling imbalance. All studies proved a point that faults manifest into failures gradually rather then abruptly. System degradation was then considered as a behavioural pattern that could be observed before complete failure occurred. This idea later supported the development of temporal learning and sequential prediction models.

Deep Reinforcement Learning (Zhou et al., 2024), Graph Neural Networks, and Transformer-based models (Arbat et al., 2022) are increasingly deployed in scheduling research around recently. Their role extends beyond prediction. They are often used to support adaptive decision-making under highly dynamic conditions. Autonomous recovery, self-healing infrastructure, and proactive orchestration methods evolved. The emphasis was on detecting problems to preventing service disruptions. These all evolution marks a gradual transition from static scheduling to intelligent operational management. Some studies focused on task execution environment. More recent works focuses on future of task execution and expected system response before a failure affects service. Despite continuous evolution numerous research gaps still exist. This provides the foundation for current research on failure-aware and mitigation-driven cloud scheduling frameworks. Current schemes focus on failure classification and accuracy improvement. Little or no attention is paid towards calculation of time estimates before degradation happen to ensure minimal wastage of resources and mitigation strategies does not

consider severity levels for all risk conditions or resource preservation requirements.

III. Proposed Approach and Methodology

The proposed FFAMSST framework attempts to address these limitations through a unified architecture that combines forecasting, scheduling and mitigation mechanisms. The framework focused on early intervention instead of delayed recovery. Prediction results were directly connected with adaptive mitigation decisions to ensure that preventive actions can be taken in time before major system degradation occur. FFAMSST also emphasized resource preservation by reducing unnecessary replication and computational overhead during mitigation. Rather than focusing only on prediction accuracy the framework aims to improve practical reliability management within dynamic cloud infrastructures.

The proposed methodology was designed to forecast machine failure in large scale cloud environments using Borg trace records and machine learning based classification models. The overall workflow consisted of data pre-processing feature engineering imbalance handling model training ensemble learning and performance evaluation. The methodology was developed to ensure that the prediction process relied only on information available before the occurrence of an actual machine failure. Lead time prediction was used to forecast the faults and provide sufficient time for mitigation module to take required action. For every machine instance the first recorded failure event was identified. The temporal distance between the current observation and the actual failure occurrence was then calculated as the lead time. Lead Time = Forecasted Time by Ensemble Model - Actual Failure Time in Dataset

The dataset sample of google cluster traces obtained from Kaggle platform has 405894 records with 34 attributes(derrick mwiti, 2021). Dataset has a sufficiently high number of workload instances. Exhaustive execution behaviours are observable in this dataset as cloud failures rarely follow a fixed pattern under varying cluster conditions. The dataset contains CPU utilization distribution, represented as percentile sequences were used to capture detailed workload variations that may not be visible through average utilization values alone. Features are further discussed in detail in experiment setup section.

To improve failure characterization a volatility-based representation of CPU i.e. CPU volatility behaviour was introduced. Statistical measures such as standard deviation variance skewness and percentile range were computed from the CPU distribution features. These measures were intended to capture unstable workload behaviour that frequently appears before machine failures in distributed cloud systems.

Parsed Resource related attributes were used to extract CPU allocation and memory allocation information. Additional engineered features cpu_gap, memory_gap, cpu_utilization, memory_utilization was then generated to represent the relationship between allocated resources and observed system usage.

Missing numerical values were handled using median based imputation while remaining undefined entries were replaced with zero values. Categorical attributes were transformed into numerical form using label encoding so that they could be processed by machine learning algorithms.

The processed dataset was partitioned into training and testing subsets using stratified sampling. Sampling was applied only to the training subset therefore the evaluation remained unbiased and closer to realistic deployment conditions. Feature scaling was performed using the Standard Scaler technique. The scaler was fitted only on the training data and later applied to the testing data to avoid information leakage during normalization.

Since machine failure instances represented the minority class within the dataset a ratio based balancing strategy was applied only on the training subset. Minority class samples were synthetically replicated until the required class ratio was achieved. The testing dataset was not changed to ensure that the final evaluation reflected realistic operational conditions. Machine learning algorithms were applied independently to train models. The models included Decision Tree, Random Forest, Gaussian Naive Bayes, Linear Support Vector Machine, XGBoost, and Cat Boost. These models were used as they represent different learning mechanisms ranging from probabilistic learning to ensemble based boosting approaches. Tree-based approaches can model nonlinear interactions, probabilistic classifiers estimate uncertainty, and margin-based approaches uses class separation. Combining these all reduces individual model bias and variance.

The algorithm for mitigation is presented here

Algorithm 1. Failure Mitigation Strategy

Initialization
Input:
J = Running Job Instance
P = Predicted Failure Probability
$T_{current}$ = Current Timestamp
$T_{failure}$ = Predicted/Actual Failure Timestamp
$H_{current}$ = Current Host

Output:
Mitigation Action A

Begin
1: Compute failure probability P (using Ensemble Predictor)
2: if $P < 0.30$ then
3: A ← Continue Normal Execution
4: return A
5: else if $0.30 \leq P < 0.60$ then
6: Enable Enhanced Monitoring
7: Create Recovery Checkpoint
8: A ← Monitor
9: else if $0.60 \leq P < 0.80$ then
10: Allocate Additional Resources
11: Create Job Replica (if task is critical nature)
12: Increase Monitoring Frequency
13: A ← Alert
14: else
15: Lead Time ← Time current as forecasted by ensemble model - Time of actual failure
16: Create Final Checkpoint
17: Search for Available Host H_{new}
18: if H_{new} is available then
19: Migrate task or job to H_{new}
20: Resume Execution from Checkpoint
21: else
22: Increase Resource Allocation
23: Activate Emergency Recovery Mode
24: end if
25: A ← Failure Mitigated
26: end if
27: return A
End

A weighted ensemble framework combined prediction of the strong performing models. Each classifier contributed to the ensemble risk prediction based on weight and probability as per equation $\text{Ensemble Probability} = \sum_{i=1}^{n} w_i * P_i$ where w_i represents the assigned model weight and P_i represents the prediction probability generated by the corresponding classifier. Probability score lies between 0 and 1. The Risk Assessment framework transformed predicted probability into operational risk category to guide scheduling decision according to algorithm 1 Failure Mitigation Strategy. If ensemble models forecast the probability below 0.30 its considered healthy and execution proceeds without intervention. Strict monitoring state exists for failure risk probability between $0.30 \leq P < 0.60$. Scheduler increased monitoring intensity and initiate precautionary replica initiations and passive checkpointing mechanisms to ensure successful executions of task on allocated machines. If probability lies between $0.60 \leq P < 0.80$ the task is classified as high risk. Additional computational resources or active replica provisioning is undertaken to reduce task failure impact on the jobs and system. If $P \geq 0.80$ framework assumes that failure is highly probable and activates mitigation procedures such as saving active checkpoints, resource reallocation, tasks and Job migrations with resumption of execution on an alternative host. This ensured actionable response rather than just forecasting as an isolated activity.

The evaluation stage involved both classification-oriented and lead-time-oriented metrics. The computation of performance metrics included Accuracy, Precision, Recall, F1 Score, ROC AUC, and PR AUC curves. Confusion metrics with comparative performance visualizations were also calculated for all models including the proposed ensemble framework. It provide a balanced predictive capability under imbalanced failure dataset.

IV. Experiment Setup

The Experimental evaluation was conducted using large-scale cloud trace dataset identified in section III. This was very useful for failure prediction. Each row represented one task instance running inside a distributed cluster system. The records contained information related to scheduling resource allocation execution behaviour and final task outcome. The target variable in the dataset is *failed*. It contains binary value i.e. 1 and 0. Value 1 indicate task failed and value 0 represents successful execution of a task. In machine learning perspective these kinds of problem belong to class of binary classification. A brief description of prominent attributes is presented below:

Features such as *collection_id*, *instance_index*, *machine_id*, and *alloc_collection_id* describe execution location and grouping of workloads inside the cluster. It provided identification information and help maintain execution context during analysis.

The dataset included multiple time related variables such as *start_time, and end_time*. These attributes are useful because cloud workloads evolve continuously during execution. Task duration can be derived using: Execution_Duration = End_time - start_time. Execution timing carried a lot of useful information regarding system instability. Tasks that show abnormal delays or irregular execution windows indicated the scheduling congestion due to resource contention inside the cluster.

Attributes *instance_events_type, collections_events_type, event_scheduler,* and *scheduling_class* describe different scheduling states of a tasks. These define operational behaviour of scheduler under dynamic workload conditions. Scheduler tries to give priority resources access to higher priority workloads while lower priority tasks may experience delay, reallocation, migrations or sometime starvations also. Such differences lead to execution failure of tasks.

Start_after_collection_ids provide dependency related information of tasks. It's essential because failure in one workload may indirectly affect all its dependent tasks scheduled in that pipeline. *Collection_name, collection_logical_name, collection_type* and *user* provide workload ownership and job level metadata. This is contextual information about workload organization.

Attribute *Constraint* define placement restrictions applied during scheduling. It helps compare requested resources with actual runtime behaviour observed during execution.

Before execution begins the attribute *resource_request* stored two values pertaining to requested CPU and requested memory so it can be parsed as per required into two columns as *cpu_resource_request and memory_resource_request*.

Actual resource consumption during workload execution and runtime usage patterns are represented by *average_usage, maximum_usage, and random_sample_usage*. *Average_usage* attribute contains two values that is of CPU and Memory so it can be parsed as per required into two columns *cpu_average_usage* and *memory_average_usage*. *Maximum_usage* attribute is parsed into two columns *cpu_maximum_usage* and *memory_maximum_usage*. *Random_sample_usage* attribute is parsed into two columns *cpu_random_sample_usage and memory_random_sample_usage*.

Assigned_memory, and *page_cache_memory* help identifies possible memory pressure conditions during execution. Memory pressure is estimated as:

Memory_Pressure = assigned_memory /available_memory. higher values indicate that workload is approaching resource saturation. Such conditions appear before execution degradation in distributed systems which is one of the key indicators for early forecasting of failures.

Processor level performance metrics such as *cycles_per_instruction* and *memory_accesses_per_instruction* capture low level execution characteristics of the workload. CPU efficiency can be approximated using:

CPU_Efficiency = 1 / cycles_per_instruction .

Lower efficiency values reflect unstable execution behaviour or inefficient workload processing.

Among all attributes workload distribution features are particularly important for early forecasting of failures.

Variables such as *cpu_usage_distribution* attribute contain eleven coarsely-spaced percentiles of the observed CPU usage during different samples: 0%ile (minimum), 10%ile, 20%ile, 30%ile, 40%ile, 50%ile, 60%ile, 70%ile, 80%ile, 90%ile, 100%ile (maximum) so parse it into eleven columns (Wilkes, 2020).

Variables such as *tail_cpu_usage_distribution* contain nine finely-spaced percentiles of the observed CPU usage during different samples: 91%ile, 92%ile, 93%ile, 94%ile, 95%ile, 96%ile, 97%ile, 98%ile, 99%ile so parse it into nine columns.

Attributes *cpu_usage_distribution*, and *tail_cpu_usage_distribution* contain sequential CPU utilization values which provide insight into workload fluctuation over time.

Since cloud failures often develop gradually sudden variation in CPU behaviour may serve as an early warning signal.

CPU volatility is estimated using the following relationship: $\text{CPU}_{\text{Volatility}} = \text{sqrt}\left(\left(\frac{1}{N}\right) * \text{sum}((x_i - \mu)^2)\right)$ where (x_i) represents CPU utilization at a specific interval and (μ) represents average utilization across the observation period. Higher volatility generally reflects unstable execution conditions.

Another characteristic of the dataset is the presence of mixed data formats. Numerical variables categorical labels and semi structured object-based attributes exist together in the same dataset. Features contain nested workload information and require additional pre-processing to be used for predictive learning.

Dataset is highly imbalanced with failure instances at 22 percent and successful events as 78 percent of dataset. $\text{Failure Ratio} = \left(\frac{\text{No of failure instances}}{\text{Total instances}}\right) = \frac{92678}{405894} = 22.83\%$ Imbalance in dataset creates complications as algorithms become biased toward the majority class during model training. Adaptive sampling techniques were used before training to get balanced dataset for appropriate training of models. Dataset provide a realistic view of workload behaviour of distributed cloud environment. It contains scheduling information, temporal patterns, resource consumption statistics, and processor level metrics. We modelled Failure forecasting, workload analysis, anomaly detection, and adaptive resource optimization using this.

The Data pre-processing steps for borg trace dataset records collected from Kaggle were pre-processed before working. Duplicate entries were removed from the dataset to avoid repeated machine events that could affect the learning process. Missing values were handled using median based imputation for numerical attributes. Remaining undefined values were replaced with zero values to preserve feature consistency during model training. Categorical variables were transformed using label encoding so that they could be processed effectively by machine learning algorithms. Further records were arranged according to instance_index and timestamp values. It helped in reconstructing the operational sequence of each machine instance over time. The Borg dataset stored CPU utilization information in the form of percentile distributions. These distributions were represented as string-based sequences stored as nested dictionary structures. Distributions were parsed and expanded into separate percentile features. Standard CPU utilization generated eleven percentile attributes ranging from p0 to p100 while tail CPU utilization contains nine additional high percentile features. Converting the distributions into explicit numerical columns allowed the models to capture finer variations in workload behaviour. To better represent unstable machine behaviour CPU volatility measures were computed from these percentile distributions. These measures included variance standard deviation skewness and percentile range statistics. The objective was to capture sudden fluctuations in processor utilization since unstable CPU behaviour is often observed before machine level failures in large scale cloud systems.

$$\text{CPU Volatility} = \sigma(\text{CPU Percentile Distribution})$$

Resource allocation information were stored in nested dictionary structures within the original dataset. These structures were parsed to extract CPU allocation and memory allocation values. For maintaining consistency across the dataset extracted attributes values were converted to numerical values. The original dictionary fields were removed to reduce unnecessary complexity only after extraction. Features engineering was applied to describe the relationship between allocated resources and observed system usage. CPU gap and memory gap variables revealed the difference between maximum observed consumption and actual requested resources. Similarly, utilization-based features were computed to estimate the operational load on each machine instance.

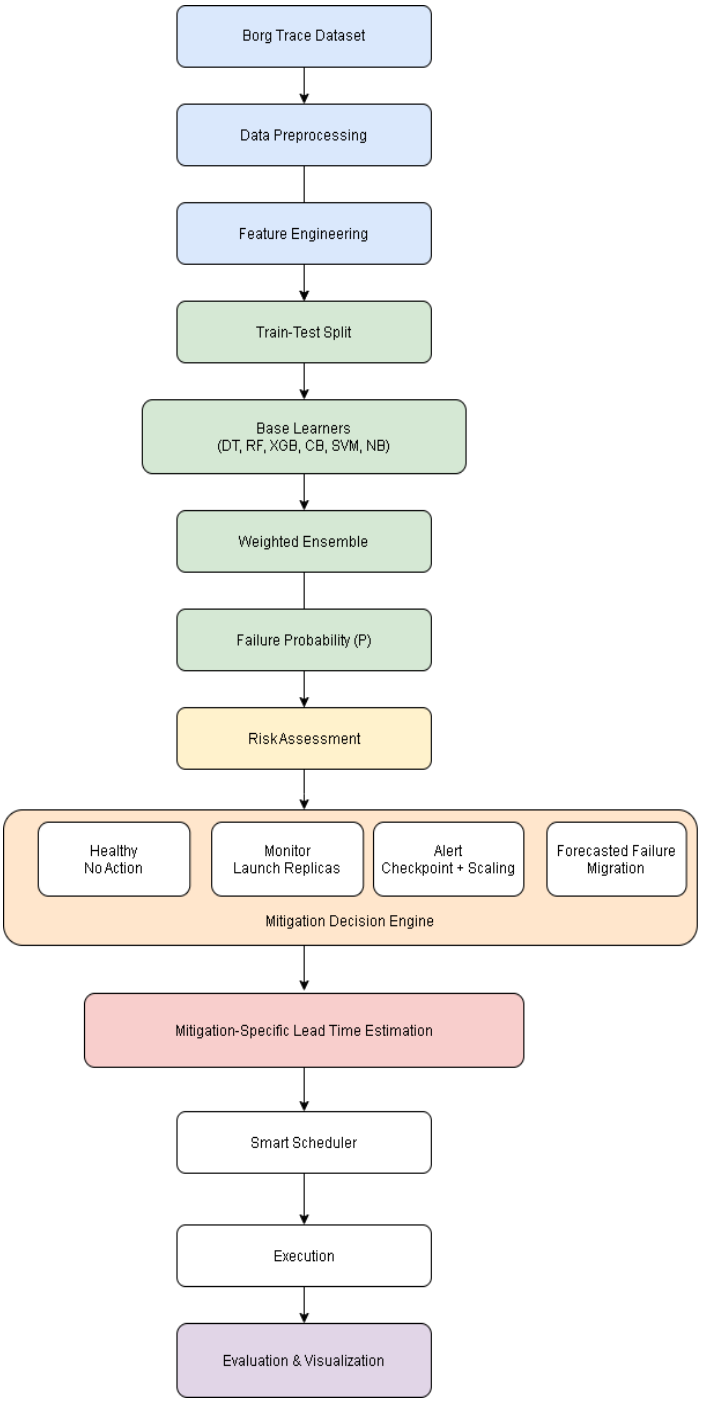

Fig. 1. Failure Forecasting Framework

$$cpu_{gap} = maximum_{usage_{cpu}} - \text{resource}_{\text{request}_{\text{cpus}}}$$

$$memory_{gap} = maximum\_usage\_memory - resource\_request\_memory$$

$$cpu_{utilization} = \frac{average\_usage\_cpu}{\text{resource\_request\_cpus}}$$

$$memory_{utilization} = \frac{Average\_usage\_memory}{resource\_request\_memory}$$

Using stratified sampling dataset was partitioned into training and testing subsets of ratio 80:20. Feature standardization was applied only on the training partition and the same scaling parameters were later applied to the testing data. This approach reduced the possibility of data leakage during pre-processing. Since the failure class remained relatively small a ratio based balancing strategy was applied only on the training data to improve minority class representation while preserving the natural distribution of the testing set.

In feature engineering steps the pre-processed dataset was supplied to multiple classification models including Decision Tree, Random Forest, Gaussian Naive Bayes, Linear Support Vector Machine, XGBoost, and CatBoost classifiers. A weighted ensemble strategy was later introduced by combining the strongest individual learners. This helped in improving prediction stability across different workload conditions and reduced dependence on a single classifier behaviour.

A lead time-based failures forecasting was achieved by calculating the difference in the fault forecasted time with the actual failure time in the dataset. Lead time was calculated as the difference between the actual failure occurrence and the time forecasted by the ensemble model. Lead Time = Forecasted Time by Ensemble Model - Actual Failure Time in Dataset.

Failures are forecasted with this lead time the more time we have in hands to apply the correct mitigation strategy for the specific case based on the priority and critical nature of the jobs.

The flow diagram is presented in Fig. 1. It mainly encompasses different stages. It starts from the raw data ➔ processed data ➔ feature engineering ➔ imbalance handling ➔ Model training➔ Ensemble Model➔ Lead time and Risk Scores calculations ➔ Mitigation Decision and action leading to a smart scheduling in cloud.

V. Results and Discussion

The dataset had highly informative resource usage and scheduling attributes. The observations about dataset and results obtained presented in graphs (Figs. 2-13) and comparative Tables (Table I & II). Performance values obtained during execution are presented in Table I. The proposed forecasting framework attained strong predictive performance for traditional as well as of ensemble learning models.

Models like Naive Bayes and Linear SVM produced slightly lower predictive performance because the cluster system events are not strictly linearly separable and does not satisfy feature independence assumptions. the tree-based methods outperform linear and probabilistic classifiers.

Random Forest model provided most stable overall performance due to its ability to model nonlinear relationships and complex feature interactions.

CatBoost and XGBoost achieved high discrimination capability because boosting based learning effectively captures subtle variations in workload behaviour and resource volatility.

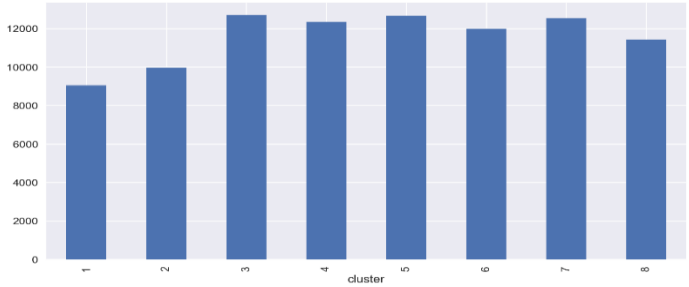

Fig. 2 Failed Jobs per Cluster

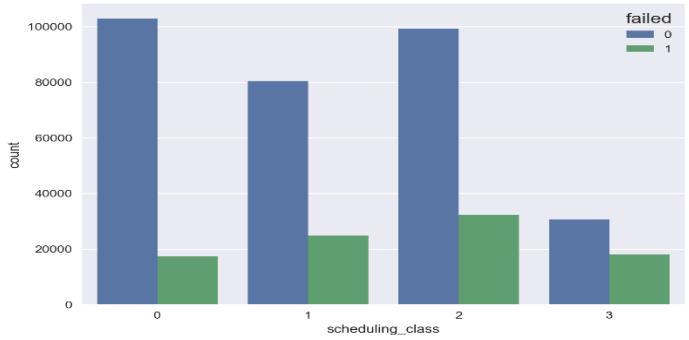

Fig. 3 Job Count per Scheduling Class

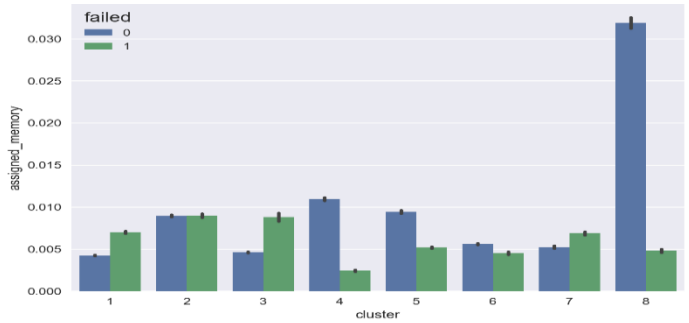

Fig. 4 Memory Usage Per Cluster

The ensemble model produced the most balanced results for all evaluation metrics. Ensemble model with weighted approach from multiple classifiers reduces variance and improves robustness under dynamic workload conditions. It stabilized lead time forecasting and reduces false alarms compared to individual models.

The distribution of lead time is presented in Fig 7. Fig. 8 shows time difference between actual and forecasted times. The proposed framework achieved reliable proactive failure forecasting with scheduling mitigation decisions for minimizing resource wastage. It enabled efficient resource reallocation and workload stabilization in large scale cluster environments.

Table I. Comparison of Different Model's Metrics

Model	Accuracy	AUC_ROC	PR_AUC	Recall	Precision	F1	TP	TN	FP	FN
LSVM	0.690856	0.746172	0.44886	0.67787	0.396497	0.500338	12565	43518	19125	5971
NB	0.771369	0.69602	0.378965	0.023522	0.486607	0.044874	436	62183	460	18100
DT	0.975659	0.990918	0.963068	0.988563	0.912186	0.94884	18324	60879	1764	212
RF	0.986363	0.999857	0.999553	0.99795	0.945367	0.970947	18498	61574	1069	38
CAT	0.989443	0.999479	0.9985	0.992069	0.962825	0.977229	18389	61933	710	147
XGB	0.995639	0.999836	0.99957	0.996224	0.984853	0.990506	18466	62359	284	70
ENSEMBLE	0.997056	0.999884	0.999651	0.993095	0.994006	0.99355	18408	62532	111	128

Table II. Summary of Results

Total_Records	Forecasted_Failures	Mean_Lead_Time	Median_Lead_Time	Max_Lead_Time	Min_Lead_Time
405894	18519	497924343911830	1872443	2677800000000	9223369917954770000

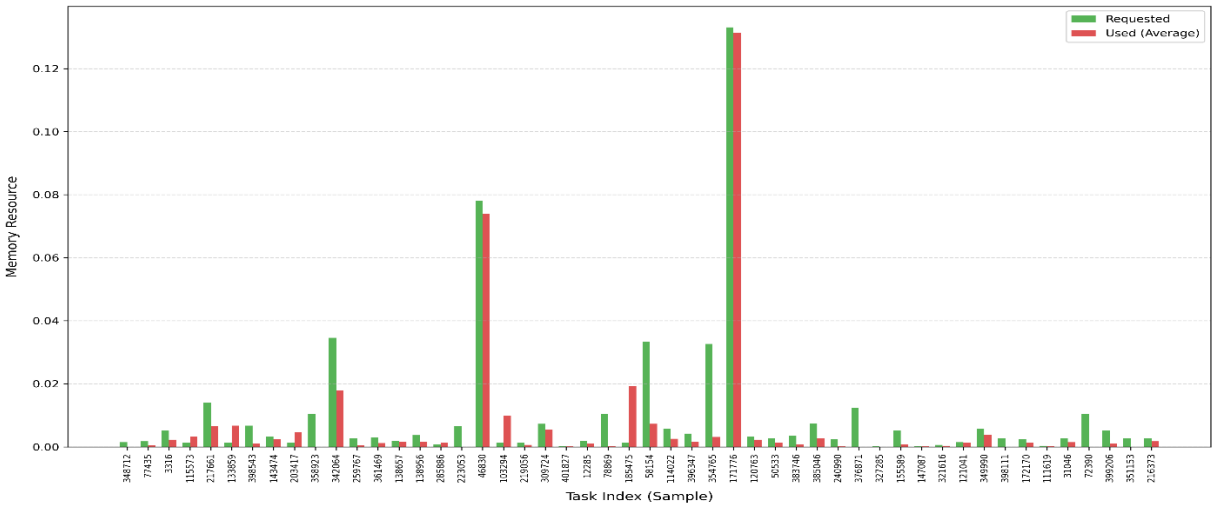

Fig. 5 Memory Usage Taskwise

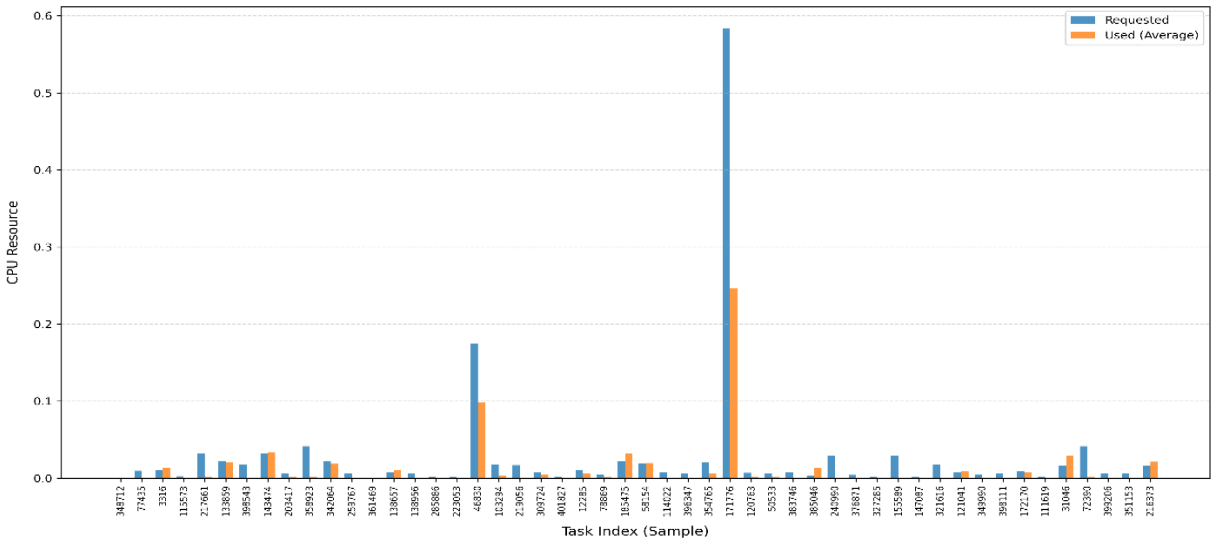

Fig. 6 CPU Usage Taskwise

Lead time identify unstable execution states way before actual failure occurrence. The different model's performance was evaluated by confusion matrix plots as in Fig 9, receiver operating characteristic assessment in Fig. 10, and comparative evaluation

of several other classification metrics are presented in Fig. 11 and 12. The analysis revealed how different learning algorithms respond to the characteristics of large-scale cloud resource data and identify the most accurate model for forecasting. Since failure prediction is inherently a risk-sensitive task, particular attention was given to balance between failure detection capability and the occurrence of classification errors.

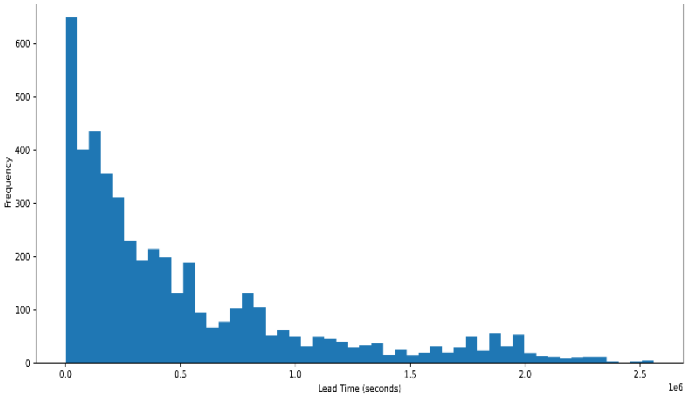

Fig. 7 Distribution of Lead Time

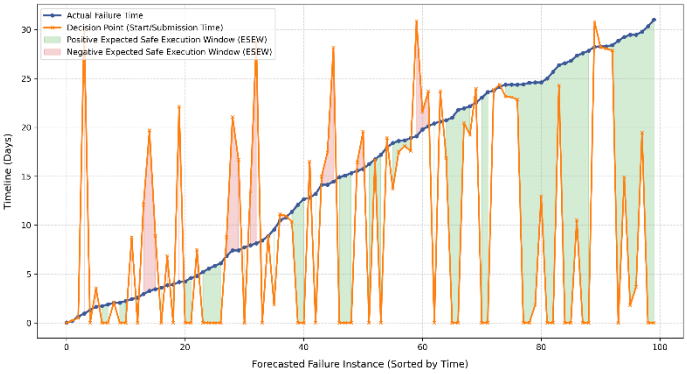

Fig. 8 Forecast versus Actual Failure Time

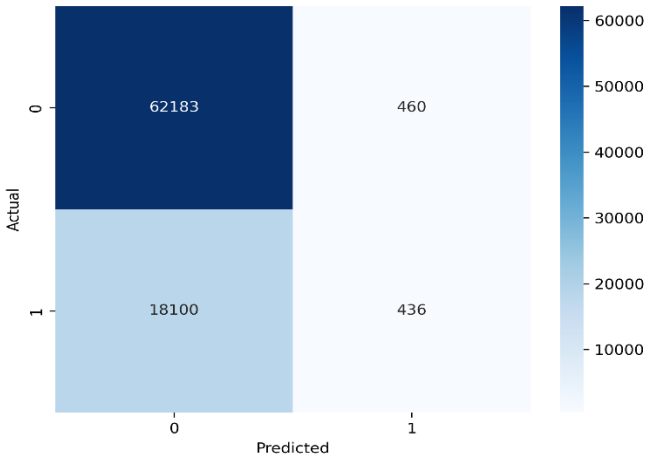

Fig. 9(a) Naive Bayes Confusion Matrix

The graphs of all model's confusion matrices are shown in Fig 9 (a)-(g) They give a clear comparison of how each model handled failure and non-failure predictions. The performance of Naive Bayes was weakest among all models, it correctly identified many failure instances, yet it produced many false negatives. This means missing a large number of actual failures due to its mechanism which assumes independence among features, whereas cloud resource variables are often related to one another. This limited its usefulness for cloud failure forecasting because undetected failures can lead to service interruptions.

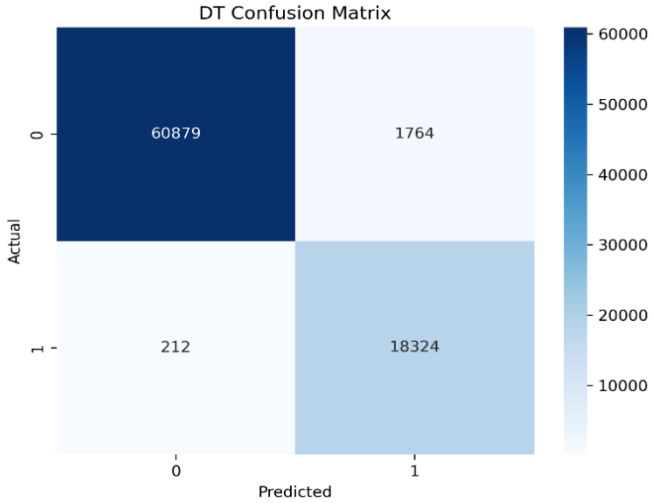

Fig. 9(b) Decision Tree Confusion Matrix

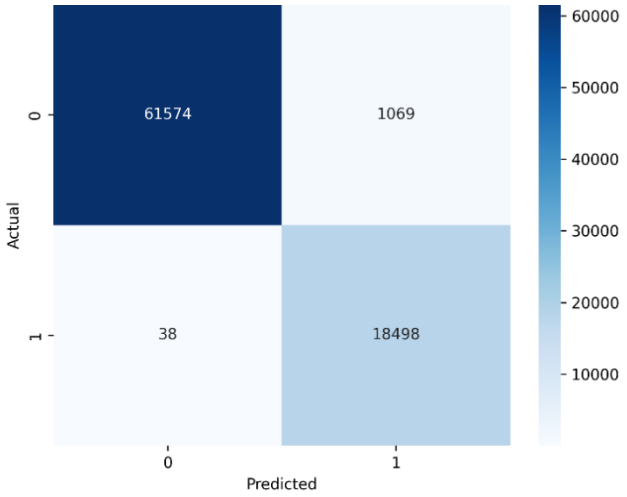

Fig. 9 (c) Random Forest Confusion Matrix

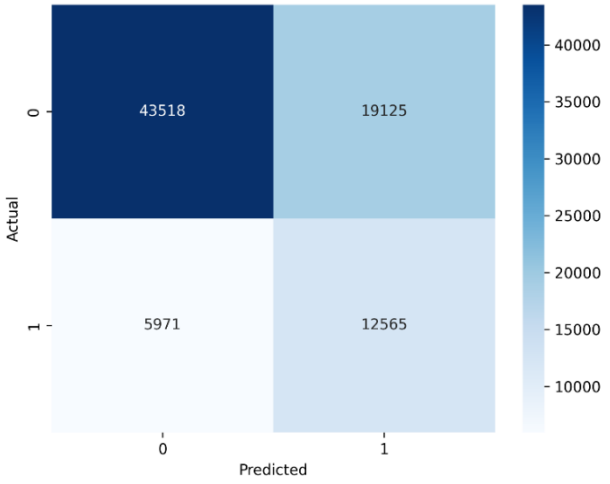

Fig. 9(d) Lightweight Support Vector Machine Confusion Matrix

The Decision Tree model correctly classified most of the observations. 60,879 instances were true positive and 1,764 were false positives. The classifier correctly identified 18,324 out of 18,536 faulty instances, with only 212 false negatives. Decision Tree was highly effective in detecting failures, while maintaining low false alarm rate.

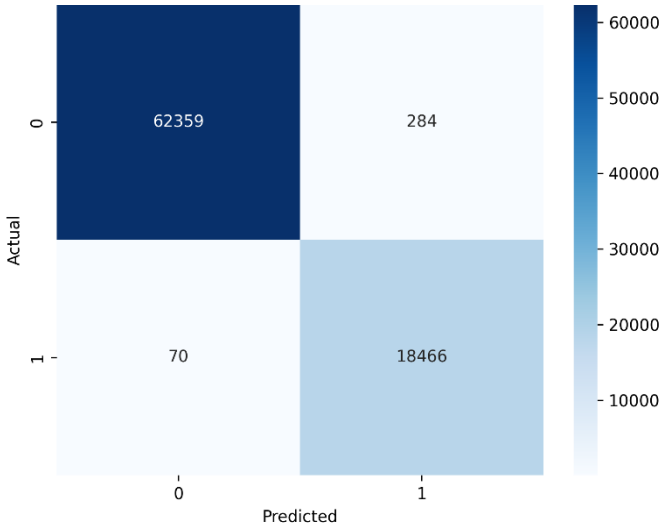

Fig. 9(e) XG Boost Confusion Matrix

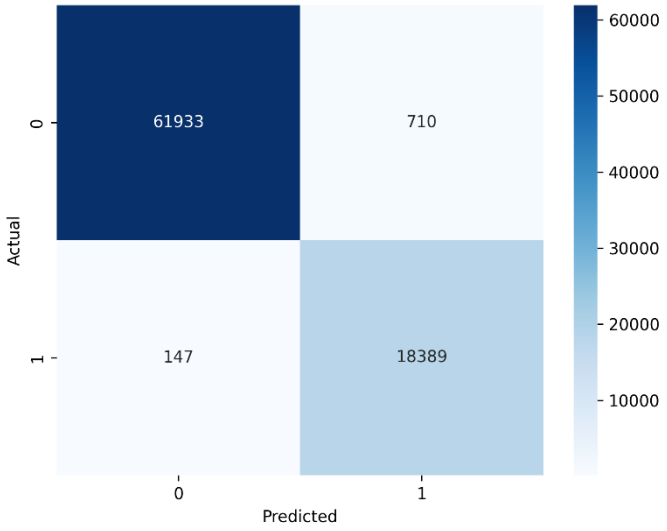

Fig. 9(f) CAT Boost Confusion Matrix

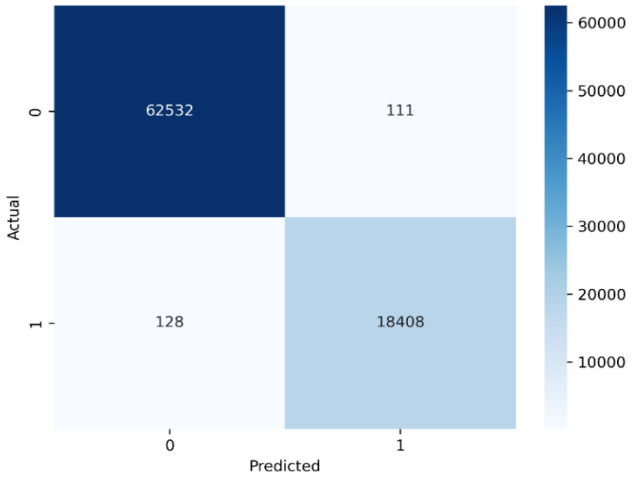

Fig. 9(g) Ensemble Confusion Matrix

Use of multiple decision trees in Random Forest which combines several learners make more stable predictions and reduce classification errors. This improved the results, false negatives dropped to 38, while false positives were reduced to 1069. XGBoost and CatBoost both performed significantly well. It has small numbers of false positives 284 and false negatives 70. Failure instances were correctly identified by these models. It means that boosting methods were able to capture the underlying patterns in the dataset more effectively than the simpler classifiers.

The LSVM model achieved strong results. However, it generated more false positives than other high-performing models. This show the aggressiveness in predicting failures. Although, this increase failure detection, it can also result in unnecessary alerts. Among all models, the proposed ensemble produced the most balanced results. It classified 128 false negatives and 111 false positives. The outcome was important because missing a failure is costlier as compared to issuing an additional warning for the low false negative and false positives count suggests that the ensemble is well suited for proactive failure management.

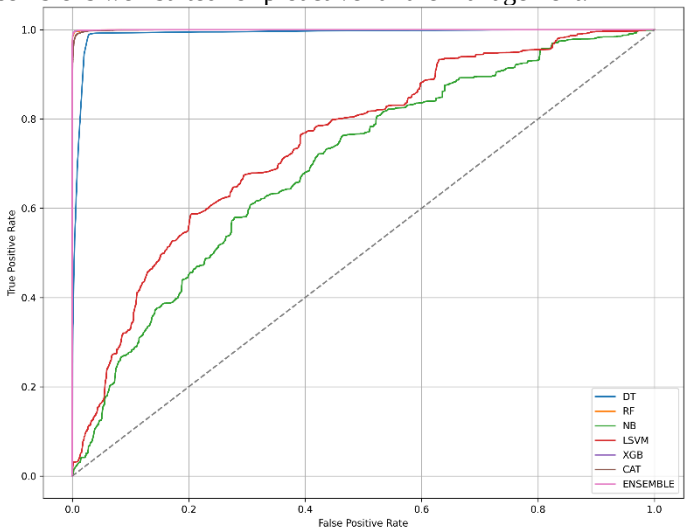

Fig. 10 ROC Comparison Curve of all models

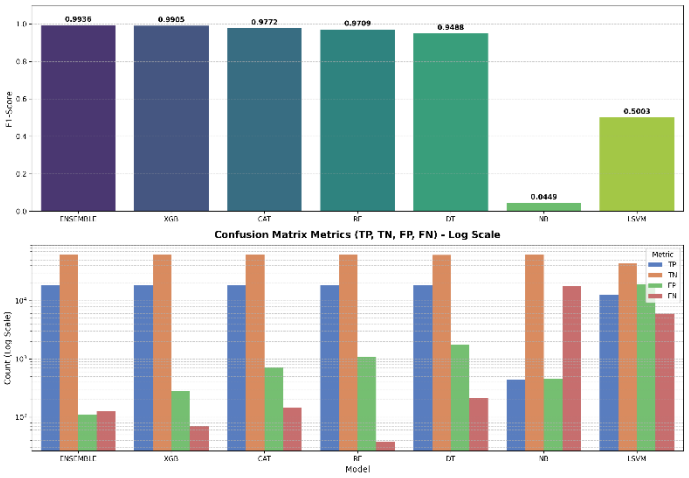

Fig. 11 F1 TP TN FP FN Comparison

The ROC curves analysis in Fig. 10 distinguish between failure and non-failure instances across different decision thresholds. A model curve closer to the upper-left corner has better discrimination capability (Fawcett, 2006). Naïve Bayes showed lowest AUC value of 0.6960. Although, this value still indicates acceptable classification performance, it was clearly lower than other models. The result is in line with the confusion matrix analysis, where a large number of failures were not detected. The Decision Tree shows AUC of 0.9909, Random Forest has 0.9998 indicating a much stronger separation between the two classes. AUC value of 0.9994 was observed for CatBoost, XGBoost. This indicated fair classification ability and almost matches to proposed ensemble approach 0.99988. The differences among these values are very small. Thus, all of them are highly effective at distinguishing between healthy and failure-prone systems.

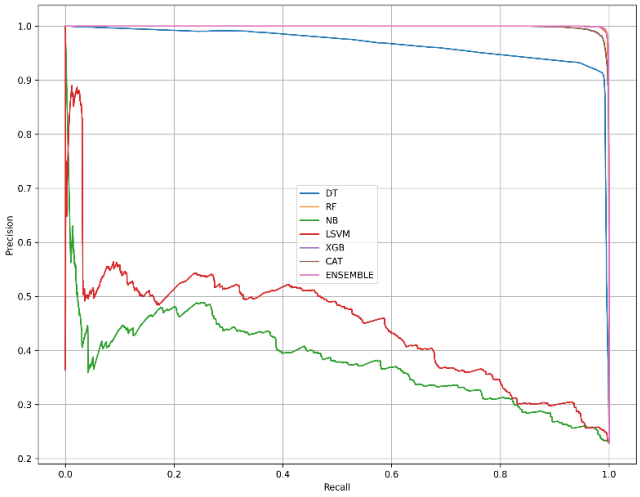

Fig. 12 PR Curve Comparison of all Models

The Comparative Performance Analysis of Accuracy, Precision, Recall, F1-score, AUC-ROC, and PR-AUC Fig 11 and Fig. 12 provided clear distinctive ability of all model's performance. Naïve Bayes achieved lowest scores across several metrics. It failed to detect many actual failures as evident in both the confusion matrix and ROC curve analyses. Decision Tree performed better than Naïve Bayes but lags behind the more advanced methods. It indicate that single-tree structure was not sufficient to fully model the behavior of the cloud resource data. Random Forest delivered strong and balanced performance across all metrics. The model achieved high recall, good precision and F1-score than that of Decision Tree and Naïve Bayes. The boosting models, XGBoost and CatBoost, produced better results as compared to other models, these models were able to learn complex relationships among the resource utilization features without introducing excessive classification errors. The proposed ensemble model achieved strong performance for all metrics like recall, precision, F1-score a perfect AUC value. All these results indicated that these models maintained high predictive quality even when dealing with class imbalance.

Finally, the results of mitigation strategy chosen by smart scheduler is presented in Fig. 13 and it mostly reflect the healthy executions as major action and some actions includes checkpointing, scaling up of resources at runtime.

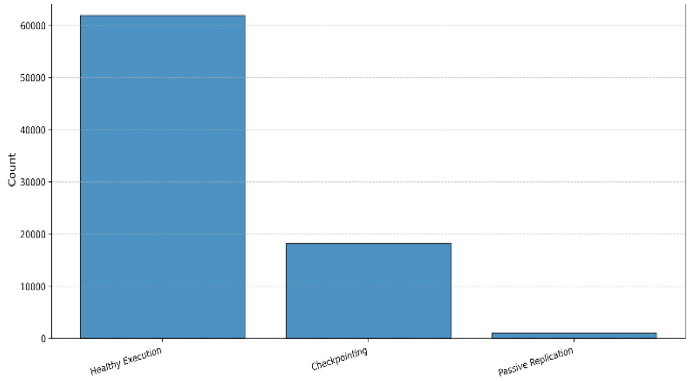

Fig. 13 Mitigation Action Distribution

VI. Conclusion and Future Scope

The proposed framework extend past the conventional failure prediction and detection approaches. It integrated prediction, warning generation, and mitigation in a single workflow. It aids system administrators and managers to take preventive measures before service disruption occurs. Decisions are derived from resource behaviour observed prior to failure, the framework remain suitable for deployment in practical cloud computing environments. Framework combine three steps. Firstly, the most consistent behaviour is used to produced risk score for each task, and estimated failure probability. Secondly, the lead time estimate indicating the expected time remaining before a failure event occurs. Lastly selecting an appropriate mitigation action based on the predicted risk level and available lead time. Possible mitigation actions include checkpoint creation, resource scaling, workload migration, replica deployment, and task recovery. Boosting mechanism captured complex and non-linear relationships present in data which are difficult for simple classifiers. XGBoost and CatBoost achieved high scores for most of the performance metrics. Reliable forecasting performance is delivered by ensemble model as it combining predictions from multiple learning algorithms which reduce the weaknesses associated with individual classifier. High accuracy, precision, recall, and F1-score were achieved with number of undetected failures remain lowest with respect to any individual models.

These characteristics are highly valuable in cloud environment where the consequences of missed failures are often more severe than those of false alarms. A false warning may introduce additional monitoring or precautionary actions, but an undetected failure can directly affect service availability and system performance. The results indicated that the proposed ensemble framework provides a practical and reliable solution for proactive failure management with improved operational stability in large-scale cloud infrastructures.

REFERENCES

Arbat, S., Jayakumar, V. K., Lee, J., Wang, W., & Kim, I. K. (2022). Wasserstein Adversarial Transformer for Cloud Workload Prediction. *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*, *36*, 12433–12439. doi: 10.1609/aaai.v36i11.21509

Caicedo, C. (2023). *A Deep Dive into the Google Cluster Workload Traces : Analyzing the Application Failure Characteristics and User Behaviors*. *FiCloud*.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. doi: 10.1145/2939672.2939785

Chen, X., Yang, L., Chen, Z., Min, G., Zheng, X., & Rong, C. (2023). Resource Allocation With Workload-Time Windows for Cloud-Based Software Services: A Deep Reinforcement Learning Approach. *IEEE Transactions on Cloud Computing*, *11*(2), 1871–1885. doi: 10.1109/TCC.2022.3169157

Cheng, M., Li, J., & Nazarian, S. (2018). DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, *2018-Janua*, 129–134. doi: 10.1109/ASPDAC.2018.8297294

Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., & Warfield, A. (2008). Remus: high availability via asynchronous virtual machine replication. *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, 161–174.

Dean, J., & Ghemawat, S. (2004, December). MapReduce: Simplified Data Processing on Large Clusters. *6th Symposium on Operating Systems Design \& Implementation (OSDI 04)*. Retrieved from https://www.usenix.org/conference/osdi-04/mapreduce-simplified-data-processing-large-clusters

Derrick, Mwiti. (2021). *Google-2019-cluster-sample*. Retrieved from https://www.kaggle.com/datasets/derrickmwiti/google-2019-cluster-sample.

Ghasemi, A., & Toroghi Haghighat, A. (2020). A multi-objective load balancing algorithm for virtual machine placement in cloud data centers based on machine learning. *Computing*, *102*(9), 2049–2072. doi: 10.1007/s00607-020-00813-w

Gokhroo, M. K. (2020). *Prediction Of Faults In Cloud Environment Using Fuzzy Approach*. *17*(4), 662–679.

Gollapalli, M., AlMetrik, M. A., AlNajrani, B. S., AlOmari, A. A., AlDawoud, S. H., AlMunsour, Y. Z., Abdulqader, M. M., & Aloup, K. M. (2022). Task Failure Prediction Using Machine Learning Techniques in the Google Cluster Trace Cloud Computing Environment. *Mathematical Modelling of Engineering Problems*, *9*(2), 545–553. doi: 10.18280/mmep.090234

Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., & Goldberg, A. (2008). *Quincy: Fair Scheduling for Distributed Computing Clusters*. *November*. Retrieved from http://www.cs.albany.edu/~jhh/courses/readings/isard.sosp09.quincy.pdf

Islam, S., Keung, J., Lee, K., & Liu, A. (2012). Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, *28*(1), 155–162. doi: https://doi.org/10.1016/j.future.2011.05.027

Iyer, G. N., & Veeravalli, B. (2011). On the resource allocation and pricing strategies in Compute Clouds using bargaining approaches. *ICON 2011 - 17th IEEE International Conference on Networks*, 147–152. doi: 10.1109/ICON.2011.6168522

Jennings, B., & Stadler, R. (2015). Resource Management in Clouds: Survey and Research Challenges. *Journal of Network and Systems Management*, *23*(3), 567–619. doi: 10.1007/s10922-014-9307-7

Le Duc, T., Leiva, R. G., Casari, P., & Östberg, P. O. (2019). Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. *ACM Computing Surveys*, *52*(5). doi: 10.1145/3341145

Li, C., Bai, J., Chen, Y., & Luo, Y. (2020). Resource and replica management strategy for optimizing financial cost and user experience in edge cloud computing system. *Information Sciences*, *516*, 33–55. doi: 10.1016/j.ins.2019.12.049

Niu, Z., Tang, S., & He, B. (2016). Gemini: An adaptive performance-fairness scheduler for data-intensive cluster computing. *Proceedings - IEEE 7th International Conference on Cloud Computing Technology and Science, CloudCom 2015*, 66–73. doi: 10.1109/CloudCom.2015.52

Rahimikhanghah, A., Tajkey, M., Rezazadeh, B., & Rahmani, A. M. (2022). Resource scheduling methods in cloud and fog computing environments: a systematic literature review. In Cluster Computing (Vol. 25, Issue 2). Springer US. doi: 10.1007/s10586-021-03467-1

Rasooli, A., & Down, D. (2011). An adaptive scheduling algorithm for dynamic heterogeneous Hadoop systems. *... of the 2011 Conference of the ...*, 30–44. Retrieved from http://xuyuanchao.ie.cnu.edu.cn/book/cloudcomputing/An Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems.pdf

Singh, S., & Chana, I. (2016). A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges. *Journal of Grid Computing*, *14*(2), 217–264. doi: 10.1007/s10723-015-9359-2

Soualhia, M., Khomh, F., & Tahar, S. (2015). Predicting Scheduling Failures in the Cloud: A Case Study with Google Clusters and Hadoop on Amazon EMR. *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*, 58–65. doi: 10.1109/HPCC-CSS-ICESS.2015.170

Wilkes, J. (2020). *{Google} cluster-usage traces v3*. Mountain View, CA, USA.

Wu, F., Wu, Q., & Tan, Y. (2015). Workflow scheduling in cloud: a survey. *Journal of Supercomputing*, *71*(9), 3373–3418. doi: 10.1007/s11227-015-1438-4

Yu, Y., Jindal, V., Yen, I. L., & Bastani, F. (2016). Integrating clustering and learning for improved workload prediction in the cloud. *IEEE International Conference on Cloud Computing, CLOUD*, *0*, 876–879. doi: 10.1109/CLOUD.2016.125

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 15–28. Retrieved from https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia

Zhou, G., Tian, W., Buyya, R., Xue, R., & Song, L. (2024). Deep reinforcement learning-based methods for resource scheduling in cloud computing: a review and future directions. *Artificial Intelligence Review*, *57*(5), 124. doi: 10.1007/s10462-024-10756-9