

Workload Failure Prediction for Data Centers

Jie Li

*Department of Computer Science
Texas Tech University
Lubbock, USA
jie.li@ttu.edu*

Rui Wang

*Department of Computer Science
Texas Tech University
Lubbock, USA
rui.wang@ttuhsc.edu*

Ghazanfar Ali

*Department of Computer Science
Texas Tech University
Lubbock, USA
Ghazanfar.Ali@ttu.edu*

Tommy Dang

*Department of Computer Science
Texas Tech University
Lubbock, USA
tommy.dang@ttu.edu*

Alan Sill

*High-Performance Computing Center
Texas Tech University
Lubbock, USA
alan.sill@ttu.edu*

Yong Chen

*Department of Computer Science
Texas Tech University
Lubbock, USA
yong.chen@ttu.edu*

Abstract—Failed workloads that consumed significant computational resources in time and space affect the efficiency of data centers significantly and thus limit the amount of scientific work that can be achieved. While the computational power has increased significantly over the years, detection and prediction of workload failures have lagged far behind and will become increasingly critical as the system scale and complexity further increase. In this study, we analyze workload traces collected from a production cluster and train machine learning models on a large amount of data sets to predict workload failures. Our prediction models consist of a queue-time model that estimates the probability of workload failures before execution and a runtime model that predicts failures at runtime. Evaluation results show that the queue-time model and runtime model can predict workload failures with a maximum precision score of 90.61% and 97.75%, respectively. By integrating the runtime model with the job scheduler, it helps reduce CPU time, memory usage by up to 16.7% and 14.53%, respectively.

Index Terms—Data Center, Failure Prediction, Predictive Analytic, Big Data, Machine Learning

I. INTRODUCTION

The scale and complexity of many data centers have significantly increased over the years. Meantime, the demand from user community for computational and storage capability has considerably increased too. This combination of increased scale of data centers and size of workloads with different requirements and characteristics has resulted in growing node and workload failures, posing a threat to the reliability, availability, and scalability (RAS) of data centers. For example, all else being equal, a system that is 1,000 times more powerful will have at least 1,000 times more components and will fail 1,000 times more often [1], resulting in a long-running job utilizing a large amount of nodes being terminated due to frequent failures. Therefore, over the past decades, various methods and algorithms were proposed to improve the system resilience and efficiency [2]–[7].

Reactive strategies, such as Checkpoint/Restart (C/R) [4], [7], are conventional approaches for fault tolerance. As an example, a reactive fault tolerance strategy for a node failure is to reschedule a workload to a new node and restart from a

specific checkpoint. However, checkpointing a job in a large-scale system could incur large I/O overhead when writing and reading workloads state [8], and takes an overhead of more than 15% of the total execution time [9], [10], which significantly impedes science productivity. As a result, researchers on failure management have found that prevention is better than cure and shifted to proactive management strategies [7], [11]–[15]. In contrast to reactive strategies, proactive strategies develop models based on the failure data in data centers to predict node or workload failures in the near future and take preventive measures to improve the RAS of data centers.

Numerous research efforts have developed node failure detection and prediction methods by utilizing temporal and/or spatial correlations of failures [16]–[19]. They usually investigate system behavior via Syslog analysis and have developed supervised and unsupervised approaches for predicting failures in data centers. A number of studies have attempted on workload-centric failure detection and prediction based on the resource usage or requested resources [20]–[23]. However, only limited amount of workload data is publicly available due to confidentiality or other reasons. In addition, analyzing and extracting insightful knowledge from massive amounts of data is daunting, given the increasing scale and complexity of data sets.

This research aims at using machine learning-based approach to predict workload failures in data centers. In particular, we investigate two months of workload traces collected from a production cluster in order to find correlation between workload attributes with exit status (including error status). We seek to train supervised learning models to predict: (1) the failure probability of a workload at queue time, and (2) the likelihood of failure over the life-span of a workload. Having the knowledge of whether jobs will likely to fail or not can be valuable for both users to be alerted of the potential failures and the resource manager (both the software and system administrators) to be proactive in preventing wasting computational resources. Consequently, the RAS and productivity of data centers can be improved in return by better

managing the workloads that are likely to fail.

We make the following contributions in this study:

- We analyze workload traces collected from a production data center and perform an extensive characterization study of workload failure rates across nodes, users and different time scales. We investigate the correlation between workload characteristics and failures, and identify the relevant factors that lead to failures.
- We apply several machine learning algorithms on our data set and train two prediction models: a *Queue-time model* and a *Runtime model*. We find that *Random Forest* achieved the best prediction performance in terms of Precision and F1 scores in both models. Experimental results show that these two models predict workload failures with a maximum Precision of 90.61% and 97.75%, respectively.
- We quantify the resource savings achieved by applying the runtime prediction model on workloads at different times. Based on the prediction results, proactive failure management (e.g., killing workloads that are predicted to fail) can achieve CPU and memory savings by up to 16.7% and 14.53%, respectively.
- We investigate the effects of training data set size to find the optimum size that can achieve acceptable prediction performance with minimum training time.

The rest of this paper is organized as follows. Section II describes background of this research, including the monitoring infrastructure, data points, and source of anomalies. In Section III, we analyze the workload data. Section IV describes the machine learning algorithms we have investigated in this research and explains our methodology. The experimental results are presented in Section V. Section VI provides an overview of related work, and we conclude this research in Section VII.

II. BACKGROUND

This research study is conducted on a production data center called *Quanah*, where scientists from all major scientific fields, such as astrophysics, computational chemistry, bioinformatics, etc., perform simulations and scientific computations. In our previous work [24], we have designed and implemented a monitoring, data collection and management infrastructure to gather workload and node metrics from the cluster in real time. The *Quanah* cluster and monitoring framework are described in the next section, followed by the analysis of sources of failures in common data centers.

A. *Quanah* Cluster

The *Quanah* cluster at High Performance Computing Center (HPCC) of Texas Tech University [25] is commissioned in early 2017 and expanded to its current size in 2019, which is comprised of 467 nodes with Intel XEON processors providing 36 cores per node. *Quanah* has a total of 16,812 cores with a benchmarked total computing power of 485 Teraflops/s and provides 2.5 petabytes storage capability. The cluster is based on Dell EMC PowerEdge™ C6320 servers, which are

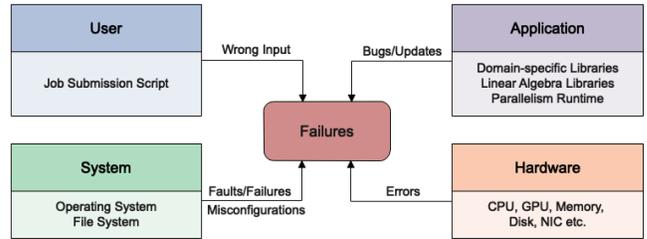


Fig. 1: Sources of Failures in Data Centers

equipped with the integrated Dell Remote Access Controller (iDRAC) [26] providing Redfish API [27] for accessing Baseboard Management Controller (BMC). The software environment is based on CentOS 7 Linux, provisioned and managed by OpenHPC, and the cluster is operated with Univa Grid Engine (UGE) [28], setting up with multiple queues, with jobs sorted by projects to meet the needs of research activities for many fields and disciplines.

B. Monitoring Infrastructure

The monitoring data in *Quanah* cluster is obtained through an “out-of-the-box” monitoring tool [24] that utilizes the Redfish API to retrieve sensor data from the BMC on each compute node and the resource manager (such as UGE and Slurm) for workload information and resource usage data. Sensor metrics and resource usage data are collected at node level in 60-second intervals, which include power usage, fan speed, CPU usage, memory usage and node-job correlations, etc. The time-series data is stored in a time-series database (e.g. InfluxDB). Workload information is derived from the UGE accounting data, which includes job submission time, start time, end time, total CPU time, integral memory usage, IO operations, etc. The workload data is stored in a MySQL database. With several performance optimizations, such as optimized database schema, using high-speed storage, concurrent processing and transmitting compressed data, our infrastructure provides near real-time analysis and visualization of user-level and node-level status.

C. Sources of Failures

In a data center cluster where computing resources are shared by different workloads submitted by users from various domains, the number of failed jobs (i.e. workloads) can be large. There are three main reasons. First, domain scientists, while skilled in their scientific fields, do not always have sufficient experience and background in computing, especially in large-scale, parallel computing. Second, diverse workloads depend on different libraries, and bugs and missing updates in dependent libraries can lead to unexpected failures. Third, data center is complex, and any mis-configuration or hardware errors can cause workload termination.

Figure 1 summarizes the high-level root cause categories in data centers, where failures are attributed to user, application, system or hardware problems.

- *Users*: insufficient resource request or wrong input in the job submission script can cause workloads to fail [29].

TABLE I: Features of Workloads

Feature	Type	Description
job_id	Numeric	Job identifier
owner	Categorical	Owner of the job
group	Categorical	The group id of the job owner
job_name	Categorical	Job name
granted_pe	Categorical	The parallel environment
hostname	Categorical	Name of the execution host
submission	Numeric	Submission time (in epoch time format)
start_time	Numeric	Start time (in epoch time format)
end_time	Numeric	End time (in epoch time format)
wallclock	Numeric	Difference between end and start time
cpu	Numeric	The CPU time usage in seconds
mem	Numeric	The integral memory usage in Gbytes ¹
io	Numeric	The amount of data transferred in Gbytes
iow	Numeric	The io wait time in seconds
maxvmem	Numeric	The maximum vmem size in bytes
slots	Numeric	The number of parallel processes
wait_time	Numeric	Difference between start and submit time
exit_status	Numeric	Exit status of the job script

¹ The sum of the amount of memory used in each time interval for the life of the job.

TABLE II: Exit Status Summary for Failed Workloads

Exit Code	Meaning	Number	Percentage
1	Miscellaneous errors	21367	58.16%
2	Missing keyword or command	3032	8.25%
7	Argument list too long	6549	17.83%
127	Command not found	528	1.44%
137	(System signal 9) Kill	190	0.52%
255	Exit status out of range	4598	12.52%
	Others	475	1.28%

Note that user interruptions, such as issuing a command like “scancel”, can interrupt a running workload and cause a failure too. However, since the effect of cancelling a running workload is obvious to the user, we do not categorize it as a source of failures.

- *Applications*: mis-configured applications increase the risk of poor performance. In addition, buggy codes, missing dependent library updates, and/or bugs can cause applications to terminate unexpectedly too.
- *Systems*: mis-configurations or failures of system resources and components may considerably affect the performance of workloads or cause failures.
- *Hardware*: hardware errors are one of the most devastating issues for data centers. Severe hardware issues can lead to the malfunction of the entire system. Events such as memory hardware errors, CPU overheating, etc., will result in workload errors and crashes.

III. WORKLOAD ANALYSIS

To predict the workload failures in data centers, it is crucial to understand the characteristics of failed workloads. In this section, we first present an overview of the workload trace, then quantitatively analyze the percentage of failures in the workloads and the computational resource consumption characteristics. After that, we further study the failure rate across the nodes, users, and different time scales.

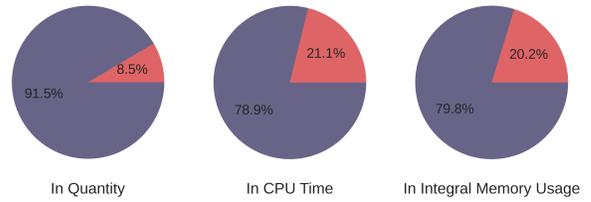


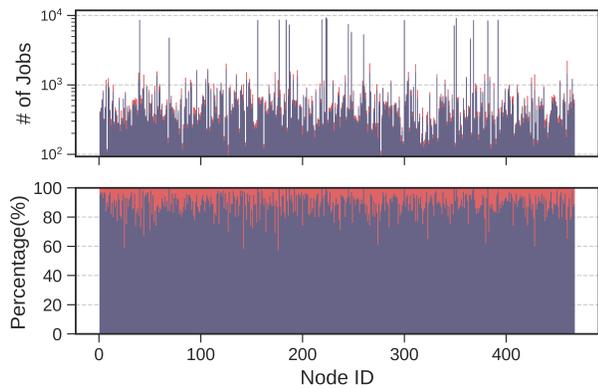
Fig. 2: Proportion and Resource Consumption Characteristics of workloads. Red indicates failed workloads and dark blue indicates successful and cancelled workloads (non-failures).

A. Workload Overview

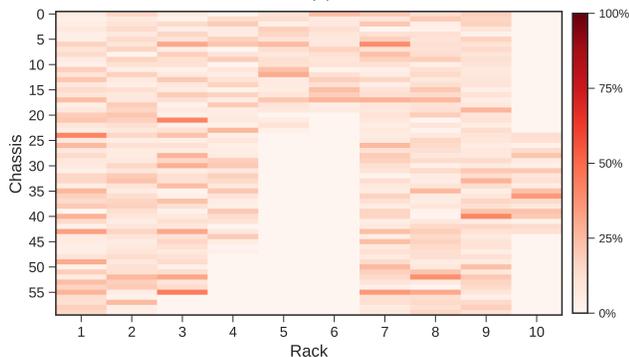
The workload trace is derived from job accounting data collected from the Quanah cluster for the period of August 1, 2020 to October 1, 2020, which contains 324,358 instances submitted by 204 unique users (i.e. owners). Notice that workloads that cannot be started on the execution host (e.g., because the user does not have a valid account on that node [30]) are recorded in the raw job accounting data, and we drop these entries because they are killed by the job scheduler for reasons that are not part of the sources of failures summarized above. Additionally, they do not consume compute resources. Table I lists 18 selected features. These features can be categorized into two groups. The first group includes categorical features, such as owner, group, and job name. The other group includes numeric features, such as CPU time usage and integral memory usage.

When a batch job exits, the scheduler (in our case UGE) generates an `exit_status` field in the job accounting data. According to the UGE documentation, a general exit status convention is defined as follows. An exit status of 0 indicates a successful workload. If the command terminates normally, the exit status is the value of the command in the job script, which is in line with normal shell conventions. In the case of the script command exits abnormally, a value of 128 is added to the value of the command. Thus, the exit status ≥ 128 can be decomposed into $128 + a$ system signal, where the system signal value can be a fatal error signal such as 6 (SIGABRT), 9 (SIGKILL).

We summarize the exit status that indicates failure in Table II. We find that the most common exit status was 1 (58.16%), indicating that there are miscellaneous errors causing the failures. The next most significant exit status is 7 (17.83%), which occurs anytime a user feeds too many arguments in the job submission script. We also notice that there are 190 (0.52%) workloads that are killed by users through system signal 9. Since we do not consider cancelled workloads as failures, we drop these 190 workloads with `exit_status` 137. In addition, we do not intent to predict exact errors, so we convert all non-zero `exit_status` to 1, representing workloads that face problems during run time. We use `exit_status` to distinguish workloads that had completed successfully (`exit_status` as 0).



(a)



(b)

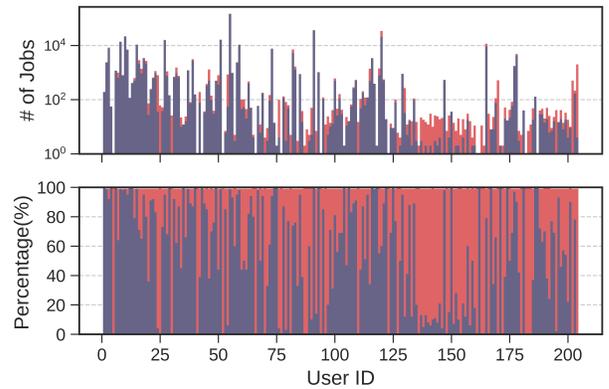
Fig. 3: Number and percentage of workload failures per node distributed by node ID (a) and by physical location (b). In sub-figure (a), red indicates failed workloads and dark blue indicates successful workloads. In sub-figure (b), the darkness of the color represents the workload failure rate (in other words, the darker the color means the higher the workload failure rate).

B. Proportion of workload failures

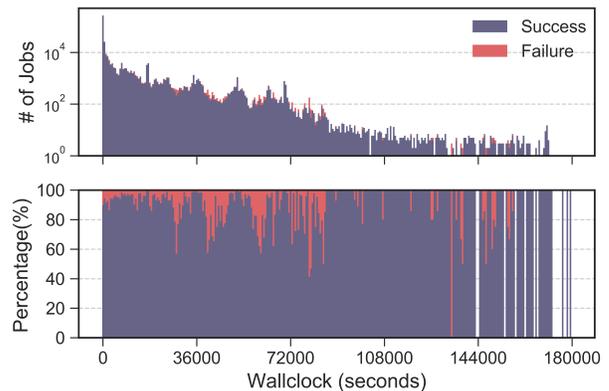
Figure 2 presents the proportion and resource consumption characteristics of workload failures. As shown in the figure, workload failure rate is 8.5% for all submitted jobs (including successful and failed workloads) in quantity. We further analyze the CPU time consumed by failed workloads and find that failed workloads cost 21.1% of the total CPU time. The proportion of CPU time for failed workloads is larger than the proportion of the number of failed workloads, indicating that the more processors a workload uses and the longer it runs, the higher the probability that this workload will fail. Additionally, we quantify the integral memory usage consumed by failed workloads. As shown in Figure 2, the wasted memory resource rate is 20.2%. All these statistics imply that failed workloads waste significant amount of computational resources and therefore degrade the system efficiency.

C. Distribution by nodes

We depict the distribution of failures across the nodes of the system by node ID and physical locations in Figure 3a



(a)



(b)

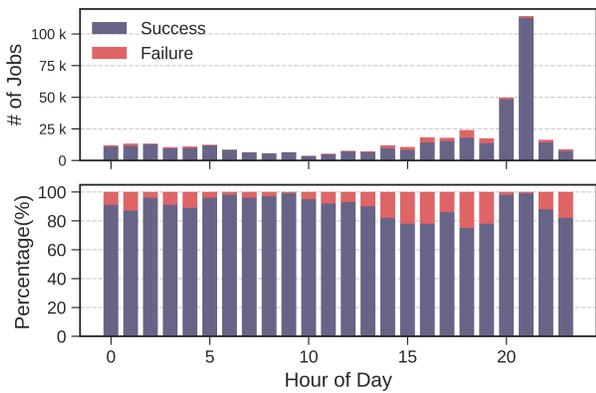
Fig. 4: Number and percentage of workload failures distributed by user ID (a) and by wallclock (b).

and Figure 3b, respectively. Figure 3a shows the total number and percentage of workload failures for each node. We first observe that about 20 nodes serve a relatively large number of workloads than the other nodes, while some nodes have more than 60% of workload failures. It is worth noting that the nodes serving a large number of workloads do not necessarily have a higher percentage of workload failures. A possible explanation is that nodes with high workload failure rate may have node-specific (hardware or operating system) vulnerabilities.

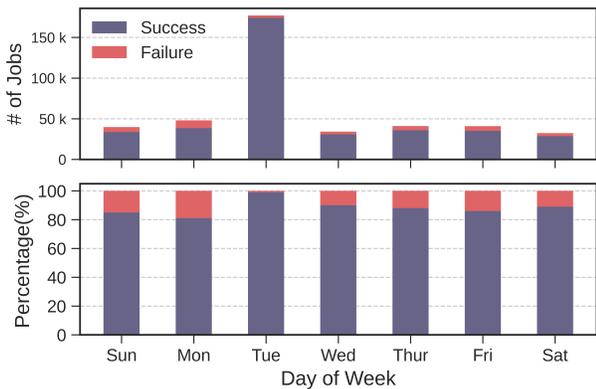
These 467 nodes in QuanaH cluster are hosted in 10 racks and each node can be uniquely addressed by rack and chassis number. Each column shown in Figure 3b represents one rack of nodes and each row represents one chassis. From Figure 3b, we observe that nodes in rack 1, 3 and 7 have relatively high workload failure rate. Since the power, temperature and connectivity of all nodes located in a rack are controlled together, problems in these areas can cause failures to occur in physical location vicinity [19].

D. Distribution by users

To find out the correlation between users and failed workloads, we plot the workload distribution by user ID, as shown in Figure 4a. The total number of jobs submitted by users



(a)



(b)

Fig. 5: Number and percentage of workload failures distributed by hour of the day (a) and by the day of the week (b).

ranges from a few to over 10,000. We observe that the failure rate of workloads per user varies significantly and those users who submit a small number of workloads have a large portion of failed workloads. These statistics suggest that users' experience in properly configuring their applications and/or requesting computational resources varies widely and that these inexperienced users contribute a large fraction of failed workloads.

E. Distribution by time

The wallclock is the actual time taken from the start to the end of a workload. On Quanah cluster, if the user does not specify a runtime in the script, the job scheduler has a default runtime limit of 172,800 seconds (48 hours) for each submitted job. We illustrate the distribution of workloads by wallclock in Figure 4b. In general, the number of workloads decreases as the wallclock increases, and there is no significant correlation between the failure rate of workloads and the number of workloads. However, we observe a reverse correlation between 24,000s and 84,000s: high workload intensities are associated with low failure rates and vice versa. In addition, the failure rate appears to be high around 144,000s as well.

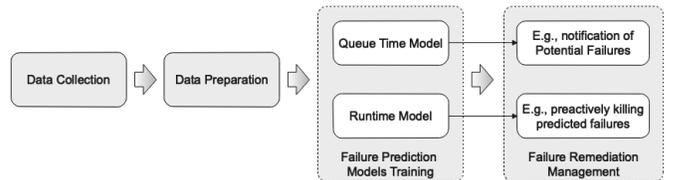


Fig. 6: Workflow of Predicting Workload Failures in Data Centers

It is commonly known that the usage pattern of data centers fluctuates with time [31]. Figure 5 categorizes the workload failures by the hour of a day and by the day of a week. We observe that there is a slight correlation between failure rate and time. During the time when the highest and the lowest number of jobs occur, the failure rate is below 5%. However, during the rest of time, the failure rate is between 10% and 22%. A possible explanation for this observation is that experienced users submit large array jobs that contribute to the majority of the workloads. Furthermore, their code is robust and their applications are well configured, resulting in low failure rates. When the workload intensity is low, the system is less vulnerable. The failure rate for each day of a week shows similar results: the highest workload volumes resulted in the lowest failure rate. However, during the rest of days of a week, the workload failure rate does not change much.

IV. METHODOLOGY

In this section, we describe the workflow of predicting workload failures in data centers. As shown in Figure 6, the workflow consists of four phases: (1) data collection: collecting metrics from data centers; (2) data preparation: preprocessing the data into a structured format and extracting features for machine learning models; (3) model training: training *Queue-time* and *Runtime* models using machine learning algorithms; (4) remediation management: applying remediation management techniques to leverage the prediction results to optimize the management of data centers. Data collection has already been discussed in Section II-B. Therefore, we focus on data preparation and model training in this research. We leave the failure remediation management and data center optimizations as near-future research work.

A. Data Preparation

1) *Preprocessing*: In our current design and implementation, the job accounting data is stored in a MySQL database; we perform a select operation with start and end times to select the data collected from August 1th, 2020 to October 1th, 2020. The data is then saved into a dataframe. In the data preprocessing phase, we convert raw features into a format more suitable for machine learning training. Specifically, we create dummy variables for the categorical variables by using one-hot encoding [32], and scale the numerical features to avoid features with high variability from having more influence in the prediction.

Another important design consideration in data preparation is to deduct irrelevant attributes and derive appropriate features from the original attributes. Irrelevant attributes add extra dimensions to the data set and can distract machine learning algorithms from achieving accurate prediction rules. On the other hand, deriving proper combinational features can boost the prediction accuracy. In our case, we drop the feature *job_id* because it is assigned by the job scheduler and does not reveal the characteristics of the workload. We derive hours of the day and days of the week from time-related features to augment the data. We also derive several numeric features from the resource usage data, such as CPU intensity and average memory usage. CPU intensity is defined as $(cpu/slots)/wallclock$, i.e., the ratio of the CPU time of a workload’s single processor to its overall wallclock (i.e. runtime). The average memory usage is defined as $mem/wallclock$, i.e., the ratio of the integral memory usage of a workload to its run time.

In addition, the collected data does not contain information about applications and libraries. To overcome this limitation, we apply Natural Language Processing (NLP) techniques to job names and identify similar job names submitted by the same user. We then assign a uniform name to these workloads as their job names. This process aims to categorize workloads that use the same libraries. An underline hypothesis of this process is that workloads with similar job names submitted by the same user tend to be the same applications and use the same libraries, differing only in parameters or parts of the code.

As discussed in Section III-A, we do not intend to predict the exact workload error, therefore the prediction is a binary classification problem (i.e., success or failure). We convert *exit_status* to 1 if the workload fails and 0 otherwise, and use this as the class label.

2) *Features Selection*: Predicting workload failures provides input for failure remediation management, where possible techniques include: 1) notifying users of potential workload failures after job submission but before execution; 2) making better scheduling decisions based on the prediction, thereby encouraging users to improve code quality and request appropriate computational resources; and 3) killing workloads that will fail before wasting too many computational resources. To this end, we plan to train two models, one for predicting pre-run failures (i.e., queue-time model) and the other for predicting runtime failures (i.e., runtime model).

Queue-time model and runtime model are trained with features available in different job states. The queue-time model is trained with categorical features such as owner, group, job name, department, etc. The runtime model uses not only categorical features but also resource usage features, such as CPU time, integral memory usage, data transferred in IO, etc. Note that in our case, resource request data, such as estimated job running time and expected maximum memory needed during runtime, are not recorded in the job accounting data, slightly limiting the available features that can be used to train the queue-time model. For other data sets that include resource request information, the queue-time model can be enhanced

and its prediction results should be more accurate.

B. Model Training

We use five classification algorithms to implement machine learning models and train them with our 324,358 instances to predict workload failures. These algorithms and the corresponding hyper-parameters are described below.

Gaussian Naive Bayes: Naive Bayes is a probabilistic machine learning algorithm based on the Bayes Theorem, which is a simple mathematical formula for calculating conditional probabilities. In our implementation, we use Gaussian Naive Bayes (GNB) (i.e., Naive Bayes extended to real-valued attributes). It is easy to implement because we only need to estimate the mean and standard deviations of the training data. Gaussian NB does not accept parameters, except for the `priors` parameter, which we use the default value of “None” in our model.

Logistic Regression: Logistic Regression (LR) is a classification algorithm for finding the relationship between features and outcome probabilities and is the most widely used machine learning algorithm in classification problems. It is relatively fast compared to other supervised classification techniques. Since we do not predict the exact value of the exit status, we use Binomial Logistic Regression. Logistic Regression does not actually have any critical hyper-parameters to tune. We set the inverse regularization parameter (i.e. `C`) to 0.1 and choose “l2” as the `penalty` parameter and “liblinear” as the `solver` parameter.

Linear Discriminant Analysis: Linear Discriminant Analysis (LDA), as the name implies, is most commonly used as a dimensionality reduction technique, but it can also be used as a classification tool by finding linear combinations of features that separate two or more classes. LDA works by calculating summary statistics, such as mean and standard deviation, of input features by class label. Predictions are performed by estimating the probability that a new instance belongs to each class label based on the values of each feature. We set the `solver` to “lsqr”, which performs best in our data set compared to other built-in solvers.

Decision Tree: Decision Tree (DT) is a predictive model that predicts value by learning decision rules inferred from data features. One of the advantages of this algorithm is that the non-linear relationship between features does not affect the performance of the tree. It can handle both categorical and numeric data. The `criterion` parameter in the DT is set to “gini” and the `splitting` parameter is set to “best”. All other parameters are kept as default.

Random Forest: Random Forest (RF) is an ensemble method that consists of a large number of individual decision trees. It uses bagging and feature randomness in the construction of each tree to create a forest of uncorrelated trees. Each individual tree in the random forest produces a class prediction and the class with the most votes will be the predicted value of the model. As with the Decision Tree, we set the `criterion` parameter to “gini” instead of “entropy”. The number of random features (i.e. `max_features`) considered in each

split is set to “sqrt”, which is usually good for classification problems. The rest of the parameters are left unchanged.

Since our data set is very large, we use the holdout method instead of the cross-validation method to save computational cost. The data set is partitioned into 65% training data, 15% validation data and 20% testing data. The training set learns the relationship between the features and the target variables (i.e. 0 for success and 1 for failure). The validation set is used to check how accurately the model defines the relationship between features and known outcomes. The testing data provides a final estimate of the model performance after the model has been trained and validated.

V. EXPERIMENTAL RESULTS

In this section, we describe the evaluation metrics we used for our experiments and present the experimental results including the performance of the machine learning algorithms described above and the potential resource savings that benefit from the prediction, followed by an evaluation of the impact of the training sizes. The models in this study are implemented in the *scikit-learn* [33] Python library.

A. Evaluation Metrics

1) *Prediction Metrics*: In order to measure the performance of ML algorithms, it is important to specify evaluation metrics. We use *recall* (i.e., true positive rate), *precision* and *F1 Score* as our measurements. Recall represents the ratio between the number of correctly predicted failed workloads to the total number of actual failures. Precision is calculated by dividing the total number of predicted failures with the number of correctly predicted failed workloads. F1 score is the weighted average of recall and precision. A higher score for these three metrics means that the model’s classification results are more accurate. These measurements are shown below:

$$recall = \frac{\# \text{ of Correctly Predicted Failures}}{\text{Total } \# \text{ of Actual Failures}} \quad (1)$$

$$precision = \frac{\# \text{ of Correctly Predicted Failures}}{\text{Total } \# \text{ of Predicted Failures}} \quad (2)$$

$$F1 \text{ Score} = \frac{2 * (recall * precision)}{recall + precision} \quad (3)$$

2) *Resource Savings Metrics*: The basic proactive failure remediation management is to simply kill workloads that are predicted to fail. This strategy is sensitive to false positive, where workloads are incorrectly predicted to fail. Killing workloads inappropriately will result in wasted resources, as the killed workloads will be restarted and run at a later time. Therefore, We define the resource saving (R_{saving}) as:

$$R_{saving} = \frac{R_s - R_w}{R_{total}}, \quad (4)$$

where R_{total} is the total resources consumed by failed and successful workloads, R_s is the resource saved by proactively killing failed workloads, and R_w is the resource wasted by killing successful workloads.

B. Failure Prediction

Table III presents the performance of the queue-time model. Specifically, we observe that Gaussian Naive Bayes (GNB) achieves the highest recall score of 99.44%; Random Forest (RF) performs the best with a precision score of 90.61% and an F1 score of 87.71%. The performance of the runtime model are shown in Table IV. Again, RF achieves the best performance with a precision of 97.75% and an F1 score of 95.91%. Although GNB achieves the highest recall score, its precision score is the lowest, indicating a low number of successful failure predictions in its total failure predictions and it predicts most successful workloads as failures. When evaluating the overall performance, we choose RF as the classification algorithm for both models.

TABLE III: Performance of Queue-Time Model

Model	recall	precision	F1 Score	Training Time(s)
GNB	99.44	15.65	27.04	3.5
LR	57.22	86.16	68.77	5.46
LDA	62.14	77.92	69.14	45.12
DT	84.02	90.33	87.06	123.96
RF	85.00	90.61	87.71	149.81

TABLE IV: Performance of Runtime Model

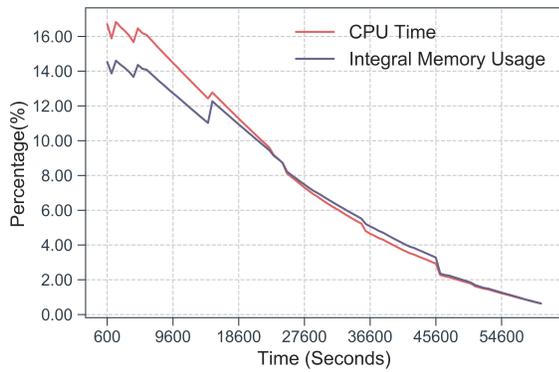
Model	recall	precision	F1 Score	Training Time(s)
GNB	99.44	15.65	27.05	5.13
LR	58.13	86.48	69.52	6.32
LDA	58.92	81.64	68.45	46.7
DT	93.92	94.57	94.24	173.08
RF	94.14	97.75	95.91	145.71

C. Resource Savings

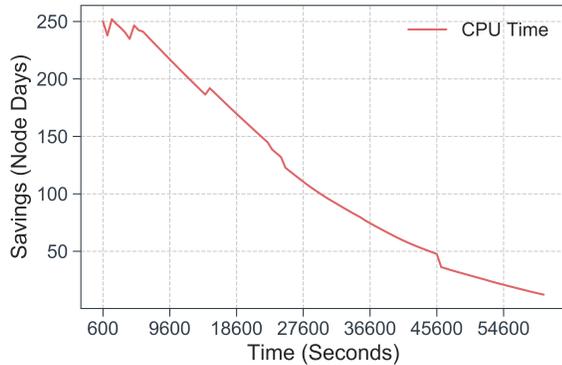
The numeric features such as CPU time and integral memory usage are only known after the workload has completed execution. This fact may raise the question of how we can estimate resource savings at different run times of a workload and use these features to predict workload failures early. To apply the run time model on a running workload, we make the following assumption: resource usage is linearly proportional to run time, so its resource usage at different times can be calculated as:

$$CRU = FRU * \frac{Time}{Wallclock}, \quad (5)$$

where CRU stands for Current Resource Usage and FRU stands for Final Resource Usage. Based on this formula, we generate a series of test data sets from the original test data (20% of 2-month workload traces in Quanah cluster, i.e., 12-day workload traces) containing synthetic resource usage at different times, and then, apply the runtime model on these data sets. Figure 7a shows the resource savings. From this figure, we observe the same pattern of savings in CPU time and integral memory usage; they both achieve the highest savings at the beginning of the time, 16.7% and 14.53%, respectively. Overall, the resource savings decrease over time, except for a few ups and downs at around 4200s and 14400s. To understand



(a)



(b)

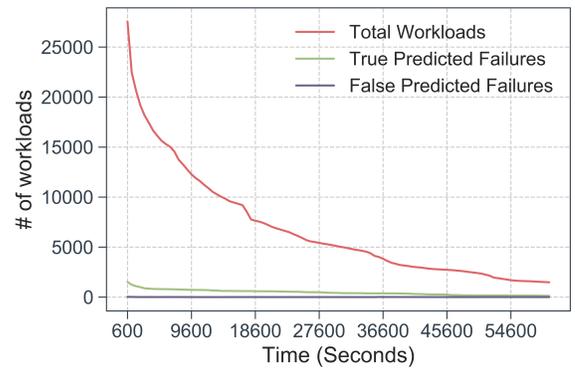
Fig. 7: Resource Savings in percentage (a) and CPU Time Savings in Node Days (b) at different times.

the resource savings, we convert the CPU time savings in *seconds* to CPU time savings in *node·days*, where the node has 36 CPUs. As shown in Figure 7b, the maximum CPU time savings is about 250 *node·days*. In other words, applying the runtime model on 12-day workloads of a 467-node cluster will help save the CPU time (and associated power consumption) of a node running for 250 days.

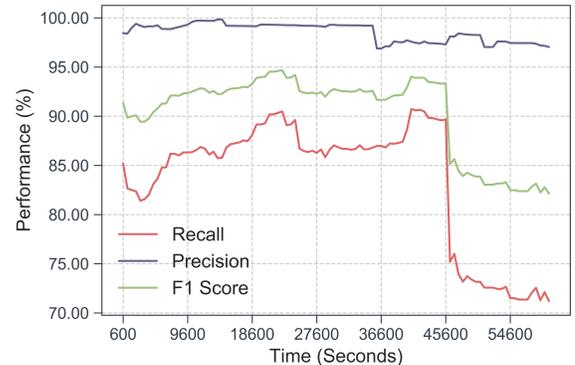
To better understand the resource savings, we plot the number of workloads and prediction performance at different times, as shown in Figure 8a and Figure 8b. Figure 8a shows that the total number of workloads participating in the failure remediation management decreases exponentially and some workloads complete before the runtime model is applied. Therefore, the resource savings that can be achieved decrease with time. Figure 8b presents the recall, precision, and F1 scores. Throughout time, the precision scores are at high values. The recall and F1 scores are decent (both scores are above 72%) although some fluctuations exist.

D. Effect of Training Size

Even though we achieve a promising prediction accuracy using Random Forest, the training time is long because of using a large amount of workload traces. As shown in Table III and Table IV, the training time in both models are about 150s.



(a)



(b)

Fig. 8: Number of workloads (a) and Prediction Performance (b) at different times.

In order to find the optimum training size that achieves a balance between prediction accuracy and training time, we build the prediction models using different training sizes in the range of 1 day to 60 days of data. As shown in Figure 9a and Figure 9b, the precision, recall and F1 scores of the queue-time model and the runtime model do not improve significantly after the training size exceeding 30 days of data. With the training size of 30 days of data, the training time shortens to 67 seconds, which is acceptable to many data centers to conduct workload failure predictions periodically.

VI. RELATED WORK

Characterizing and quantifying failures in data centers are invaluable for system administrators to understand the behavior of the systems and thus develop strategies to improve the RAS of the systems. Many prior works have investigated failures on large-scale systems [16], [17], [19], [20], [31], [34], [35]. For example, Fadishei et al. [20] analyzed workload traces in grid environment and discovered correlations between failure characteristics and performance metrics. Schroeder et al. [31] examined statistics on failure data collected at two large HPC sites and discovered temporal and spatial correlations of failures. Zheng et al. [34] presented a co-analysis of RAS and job logs that helps in understanding failure patterns

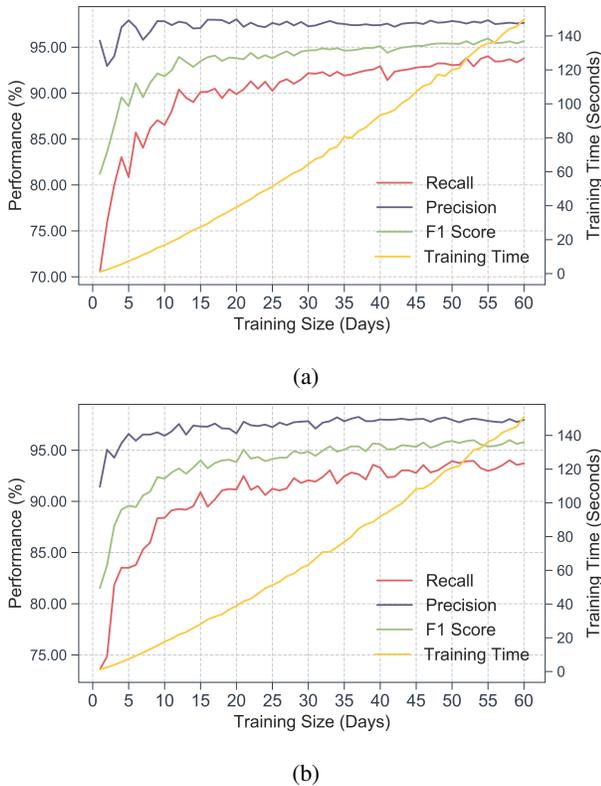


Fig. 9: Training Size vs. Prediction Performance of Queue-time Model (a) and Runtime Model (b).

and system/user behavior. There are also studies that looked specifically into the reliability of particular component such as DARMs, disks and GPUs [36]–[38].

Considering the failure characteristics and the correlations between failures and job types, performance metrics and components, several studies investigated machine learning models to predict failures on large-scale systems [21], [22], [39], [40]. Fu et al. [39] proposed a hybrid failure detection framework using one-class and two-class support vector machines (SVM). Chen et al. [21] proposed a prediction method based on Recurrent Neural Network (RNN) that predicts application failure in cloud using the Google cluster workload traces. Tariqul et al. [22] developed a similar approach like Chen’s by using Long Short-Term Memory Network (LSTM). Many of the proposed approaches are limited to certain performance metrics, such as studies based on Google cluster workload traces [21], [22], or are limited to certain components of the system, such as studies focused on GPUs [40].

The drawback of these mentioned approaches is that they ignore the human factors that lead to failures. As shown in Section III-D and Section III-E, there are correlations between failures and user behavior. A well-trained and experienced user can potentially produce less failure jobs. The proposed approach in this work considers not only performance metrics, but also user behavior in the prediction models. In addition, the proposed approach does not rely on complex system

logs collection and analysis; it utilizes job accounting data that is available in all resource managers. Therefore, the prediction models and failure remediation mechanisms (e.g. killing predicted failures) are easier to integrate into resource managers.

VII. CONCLUSIONS AND FUTURE WORK

In this study, we have analyzed two months of job accounting data collected from a production data center and found that failed workloads accounted for 8.5% of total workloads, consumed 21.1% of the total CPU time and 20.2% of the integral memory usage. In addition, we have quantified the workload failure rates across nodes, users, and different time scales, and we have analyzed the correlation between them. Based on the comprehensive understanding of workload traces, we develop two prediction models (queue-time model and runtime model) with five machine learning algorithms and have found that Random Forest performed the best with 90.61% and 97.75% precision scores, respectively. We further explored the training size and its impact on prediction performance and training time, and we concluded that 30 days of job data is the optimal training size, with 67 seconds of training time for our data sets. Our experimental results show that the workload failure prediction model can help save CPU time and integrated memory usage by up to 16.7% and 14.53%, respectively.

Nevertheless, our study can be further improved in several aspects. First, due to the lack of resource usage data for workloads at different runtimes, we had to create synthetic data to quantify the resource savings gained from the runtime model. This approach may not be representative of all situations and the accuracy of the predictions may not be as high as expected. Second, because resource request information is an important factor in predicting workload failure, the lack of this feature prevented our model from achieving more accurate predictions. Third, the prediction models only predict the probability of workload failure. Even though we have achieved promising performance, we cannot infer the causality of workload failure based on the available data since correlation does not imply causality. To further support causality identification, we plan to develop a provenance based approach for failure predictions in the future.

In large-scale data centers, where workload failures become the norm, proactive failure management is critical to improve system reliability, availability, and scalability. In future work, we plan to improve the prediction by adding more features in the training data, such as hardware monitoring metrics and system logs, and explore other machine learning algorithms, such as LSTM. In addition, understanding the causality of workload failures is important for both system administrators and users. We hope to conduct causal inference studies when the detailed provenance is available. Moreover, failure-aware resource scheduling is also a promising research direction and deserves further studies.

REFERENCES

- [1] F. Cappello, G. Ai, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward exascale resilience: 2014 update," *Supercomputing Frontiers and Innovations: an International Journal*, vol. 1, no. 1, pp. 5–28, 2014.
- [2] G. Candea, A. B. Brown, A. Fox, and D. Patterson, "Recovery-oriented computing: Building multitier dependability," *Computer*, vol. 37, no. 11, pp. 60–67, 2004.
- [3] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot—a technique for cheap recovery," *arXiv preprint cs/0406005*, 2004.
- [4] P. H. Hargrove and J. C. Duell, "Berkeley lab checkpoint/restart (blcr) for linux clusters," in *Journal of Physics: Conference Series*, vol. 46, no. 1. IOP Publishing, 2006, p. 067.
- [5] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 732–749, 2011.
- [6] G. Aupy, M. Shantharam, A. Benoit, Y. Robert, and P. Raghavan, "Co-scheduling algorithms for high-throughput workload execution," *Journal of Scheduling*, vol. 19, no. 6, pp. 627–640, 2016.
- [7] M. Rodríguez-Pascual, J. Cao, J. A. Morfíño, G. Cooperman, and R. Mayo-García, "Job migration in hpc clusters by means of checkpoint/restart," *The Journal of Supercomputing*, vol. 75, no. 10, pp. 6517–6541, 2019.
- [8] R. Garg, T. Patel, G. Cooperman, and D. Tiwari, "Shiraz: Exploiting system reliability and application resilience characteristics to improve large scale system throughput," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 83–94.
- [9] E. N. Elnozahy and J. S. Plank, "Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 97–108, 2004.
- [10] F. Cappello, "Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities," *The International Journal of High Performance Computing Applications*, vol. 23, no. 3, pp. 212–226, 2009.
- [11] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam, "Critical event prediction for proactive management in large-scale computer clusters," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 426–435.
- [12] P. Yalagandula, S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems," in *WORLDS*, 2004.
- [13] J. W. Mickens and B. D. Noble, "Exploiting availability prediction in distributed systems," in *NSDI*, vol. 6, 2006, pp. 73–86.
- [14] A. Nukada, H. Takizawa, and S. Matsuoka, "Nvcr: A transparent checkpoint-restart library for nvidia cuda," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. IEEE, 2011, pp. 104–113.
- [15] A. Rezaei, G. Coviello, C.-H. Li, S. Chakradhar, and F. Mueller, "Snapify: Capturing snapshots of offload applications on xeon phi many-core processors," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, 2014, pp. 1–12.
- [16] N. El-Sayed and B. Schroeder, "Reading between the lines of failure logs: Understanding how hpc systems fail," in *2013 43rd annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2013, pp. 1–12.
- [17] S. Ghiasvand, F. M. Ciorba, R. Tschüter, and W. E. Nagel, "Lessons learned from spatial and temporal correlation of node failures in high performance computers," in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. IEEE, 2016, pp. 377–381.
- [18] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," *IEICE Transactions on Communications*, 2018.
- [19] S. Ghiasvand and F. M. Ciorba, "Anomaly detection in high performance computers: A vicinity perspective," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2019, pp. 112–120.
- [20] H. Fadishei, H. Saadatfar, and H. Deldari, "Job failure prediction in grid environment based on workload characteristics," in *2009 14th International CSI Computer Conference*. IEEE, 2009, pp. 329–334.
- [21] X. Chen, C.-D. Lu, and K. Pattabiraman, "Failure prediction of jobs in compute clouds: A google cluster case study," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. IEEE, 2014, pp. 341–346.
- [22] T. Islam and D. Manivannan, "Predicting application failure in cloud: A machine learning approach," in *2017 IEEE International Conference on Cognitive Computing (ICCC)*. IEEE, 2017, pp. 24–31.
- [23] D. Andresen, W. Hsu, H. Yang, and A. Okanlawon, "Machine learning for predictive analytics of compute cluster jobs," *arXiv preprint arXiv:1806.01116*, 2018.
- [24] J. Li, G. Ali, N. Nguyen, J. Hass, A. Sill, T. Dang, and Y. Chen, "Monster: An out-of-the-box monitoring tool for high performance computing systems," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2020, pp. 119–129.
- [25] HPCC. (2021) High Performance Computing Center. [Online]. Available: <http://www.depts.ttu.edu/hpcc/>
- [26] D. Technologies. (2021) Integrated Dell Remote Access Controller (iDRAC). [Online]. Available: <https://www.delltechnologies.com/en-us/solutions/openmanage/idrac.htm>
- [27] DMTF. (2021) DMTF's Redfish®. [Online]. Available: <https://www.dmtf.org/standards/redfish>
- [28] U. G. Engine. (2020) Univa Grid Engine. [Online]. Available: <https://www.univa.com/>
- [29] H. Li, D. Groep, L. Wolters, and J. Templon, "Job failure analysis and its implications in a large-scale production grid," in *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*. IEEE, 2006, pp. 27–27.
- [30] G. Engine. (2010) Grid engine Man Pages. [Online]. Available: <http://gridscheduler.sourceforge.net/htmlman/htmlman5/accounting.html>
- [31] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, 2009.
- [32] Wikipedia. (2021) Dummy variable (statistics). [Online]. Available: [https://en.wikipedia.org/wiki/Dummy_variable_\(statistics\)](https://en.wikipedia.org/wiki/Dummy_variable_(statistics))
- [33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
- [34] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner, "Co-analysis of ras log and job log on blue gene/p," in *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2011, pp. 840–851.
- [35] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 610–621.
- [36] A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 111–122, 2012.
- [37] V. Sridharan, J. Stearley, N. DeBardleben, S. Blanchard, and S. Gurusurthy, "Feng shui of supercomputer memory positional effects in dram and sram faults," in *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2013, pp. 1–11.
- [38] B. Nie, J. Xue, S. Gupta, C. Engelmann, E. Smirni, and D. Tiwari, "Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities," in *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2017, pp. 22–31.
- [39] S. Fu, J. Liu, and H. Pannu, "A hybrid anomaly detection framework in cloud computing using one-class and two-class support vector machines," in *International conference on advanced data mining and applications*. Springer, 2012, pp. 726–738.
- [40] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine learning models for gpu error prediction in a large scale hpc system," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 95–106.