Predicting and Mitigating Jobs Failures in Big Data Clusters

Andrea Rosà
Faculty of Informatics
Università della Svizzera Italiana
Lugano, Switzerland
Email: andrea.rosa@usi.ch

Lydia Y. Chen
Cloud Server Technologies Group
IBM Research Lab Zurich
Rüschlikon, Switzerland
Email: yic@zurich.ibm.com

Walter Binder
Faculty of Informatics
Università della Svizzera Italiana
Lugano, Switzerland
Email: walter.binder@usi.ch

***Abstract*—In large-scale datacenters, software and hardware failures are frequent, resulting in failures of job executions that may cause significant resource waste and performance deterioration. To proactively minimize the resource inefficiency due to job failures, it is important to identify and predict failed jobs using key job attributes. However, so far, prevailing research on datacenter workload characterization has overlooked job failures, including their patterns, root causes, and impact. In this paper, we aim at developing prediction models and mitigation policies for unsuccessful jobs, so as to reduce the resource waste in big datacenters. In particular, we base our analysis on Google cluster traces, consisting of a large number of big-data jobs with a high task fanout. We first identify the time-varying patterns of failed jobs and the contributing system features. Based on our characterization study, we develop an on-line predictive model for job failures by applying various statistical learning techniques, namely Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), and Logistic Regression (LR). Furthermore, we propose a delay-based mitigation policy, which after a certain grace period proactively terminates the execution of jobs that are predicted to fail. The particular objective of postponing job terminations is to strike a good tradeoff between resource waste and false prediction of successful jobs. Our evaluation results show that the proposed method is able to significantly reduce the resource waste by 41.9% on average, and keep false terminations of jobs low, i.e., only 1%.**

I. Introduction

Due to ever increasing scale of systems and the complexity of applications, failures in either software or hardware are more norm than an exception in today's large-scale datacenters [1]. Consequently, failures of big-data applications that feature high task fanout prevail [2] and potentially turn into critical performance impediment. Tasks of failed jobs can easily use computational and storage resources for a significant amount of time and lead to a waste of resources, e.g., CPU, volatile memory, disk space, and may block the execution of other jobs. Using publicly available big-data cluster traces [3] as a motivating case for the negative impact of failed jobs, Figure 1 summarizes the percentage of computational time and resources utilized by successful and failed jobs. The daunting observation is that an overwhelming amount of resources are used by jobs which eventually fail. To increase the resource efficiency and the performance of datacenters executing big-data jobs, it is essential to minimize the resource waste due to failed jobs.

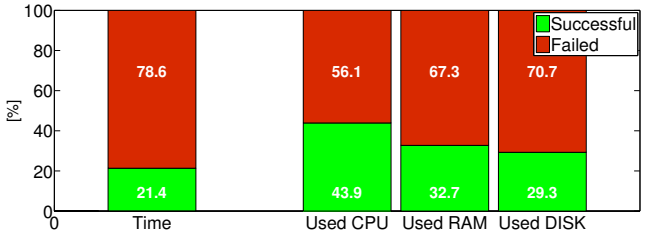


Fig. 1. Distribution of computational time and used resources among successful and failed jobs in the Google cluster [3].

A deep understanding of job failures is the first step towards mitigating their negative impact on system resources and application performance. However, while prior work addresses dependability issues in datacenters either from the perspective of hardware reliability [4], [5] or software bugs [6], [7], little is known about application job failures in terms of patterns and root causes. Recently, a few studies based on Google traces [8], [9] conduct an in-depth analysis to characterize the resource demands of big-data workloads, but the dependability issues are overlooked.

Indeed, the executions of big-data jobs in large-scale datacenters are highly complex, due to the underlying heterogeneous hardware, intricate dependencies among tasks, diversified job priorities, and sophisticated job scheduler. Jobs typically have multiple tasks, which can have disparate resource demands and be executed on dissimilar hardware. Individual tasks may not be executed successfully and thus require re-execution, depending on the scheduling policies and overall system dynamics. For example, the priority scheduler can evict tasks from lower priorities upon arrivals of higher priorities, when a shortage of resources occurs to the system [10]. Moreover, job arrivals in such datacenters exhibit strong time-variability [8] and this further exacerbates the difficulty of predicting the outcome of job executions.

In this paper, we aim at answering the following challenging question: how can we predict failed jobs accurately as soon as jobs arrive and further minimize resource waste proactively? Prior to developing systematic predicting methods and practical mitigation policies, we first conduct a characterization study that identifies the key system features attributing to failed

jobs. In particular, we employ machine learning techniques, namely Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Logistic Regression (LR), and some variations of these techniques. To address the characteristics of time-varying system dynamics, we leverage on-line training based on historical data in a moving window fashion. We are able to predict job outcomes upon their arrivals using features computed from the historical data, and further apply delay-based mitigation policies that proactively terminate jobs based on the prediction results and after a certain delay.

Our study is based on Google cluster traces [3], containing workload information on 29 days. We identify both static and system features upon every job arrival. The static features are composed of statistics related to jobs, such as the number of tasks and the amount of requested resources across tasks. Due to the multi-task nature of jobs, we consider both the average and the standard deviation of each feature across tasks of the same job. The system features capture the system dynamics across all priorities, including arrival rates and the number of tasks from different priorities. We use the first 14 days as training set, by which we identify the best machine learning technique, i.e., LDA on expanded basis, and the optimal duration for the moving window. As for delay-based mitigation policies, we first pre-select a set of delay thresholds to terminate jobs and search for the optimal value during the training phase. The main objective of waiting for a grace period after obtaining failure prediction is to minimize false prediction of successful jobs, i.e., the false negatives. Evaluation results on the testing set, i.e., the remainder of the traces, shows that the proposed prediction model and mitigation policy are able to achieve an average resource saving of 41.9% among all cluster resources, with 1.05% of false terminations of successful jobs.

Our scientific contributions are threefold. First, to the best of our knowledge, we provide one of the first studies on failed jobs of modern big-data applications using field data. We identify and characterize a set of critical system features, which can strengthen the accuracy of prediction, in comparison with a model using only static features. Secondly, we incorporate key system features into statistical learning models, particularly addressing time-variability of different priorities, and predict job failures upon arrival. Thirdly, we provide a complete methodology for identifying, predicting, and mitigating job failures in large-scale datacenters. Our solution strikes a good balance among prediction accuracy, resource waste, and unnecessary job terminations.

Section II provides a basic description of the used data set, followed by a characterization study on job failures and key system features in Section III. We detail our online prediction method and the design of the mitigation policy in Section IV. In Section V, we present extensive evaluation results, followed by a discussion on related work in Section VI. Section VII concludes with a summary of our work.

II. Data and System Description

Our analysis is based on Google cluster traces [3], which represents a rich heterogeneous workload mix executing on a large heterogenous cluster for 29 days. The trace format can be found in the trace specifications [11]. Here, we first provide the relevant description about job failures at Google datacenter and our methodology to sanitize the data. Last, we also summarize the basic statistics regarding successful and failed jobs.

A. Trace Description

The Google traces contain 672074 jobs, submitted by users from the midnight of May 1^{st}, 2011 to the midnight of May 29^{th}, 2011. Each job contains multiple tasks, ranging from 1 to more than 90000. Upon job submission, users also specify the resource requirements of its tasks, in terms of CPU, volatile memory, and disk. CPU expresses the maximum number of cores that tasks can use, while RAM and DISK specify the amount of volatile and mass memory, respectively. These amounts may be further leveraged by the scheduler, and tasks' actual usages can be greater or smaller than the requested amount. Moreover, a priority is assigned to each task, ranging from 0 to 11. Higher values of priority are associated with "production" tasks. In summary, both requested resources and priorities are defined at the task level.

As soon as jobs enter the cluster, their tasks are considered *submitted* to the system and wait for a scheduling decision. Later, tasks are *scheduled* on machines and begin their execution. Tasks *complete* their execution with one of the following types: `finish`, `eviction`, `fail`, and `kill`. We consider the last three types as *failures*. After a failure, tasks can either go through the same life cycle (being submitted again) or leave the system permanently. Moreover, job terminations are also classified into those four types. The distribution of job types is given in Table I.

TABLE I
DISTRIBUTION OF JOB OUTCOMES IN THE ENTIRE TRACE: ABSOLUTE NUMBERS AND PERCENTAGES.

	Failed			Successful	
Type	`Eviction`	`Kill`	`Fail`	`Finish`	Total
No.	18	268817	10046	385461	664342
%	∼0	40.5	1.5	58.0	100

The aforementioned information is scattered in multiple tables, each of which serves different purpose and covers different job, tasks and machines attributes. Tables are split into several files by timestamps. In particular, we focus on the Task and Job events tables, from which we extract key features for the proposed predictive model. Each job has unique identifier (Job ID) and each task has also an unique identifier that specifies its Job ID and the index number within the job. We use their unique identifiers to link Task and Job events tables.

In particular, the two tables store submissions, schedulings and terminations experienced by each task or job during its life, the timestamps when such events take place, the resources

TABLE II
MEAN STATISTICS PER FAILED AND SUCCESSFUL JOB: NUMBER OF TASKS, AVERAGE (AVG) AND STANDARD DEVIATION (STD) OF REQUESTED RESOURCES AND PRIORITIES PER JOB. AVG AND STD ARE COMPUTED OVER TASKS BELONGING TO THE SAME JOB.

	N. tasks	CPU		RAM		DISK		Priority	
	AVG	AVG	STD	AVG	STD	AVG	STD	AVG	STD
All	37.61	$2.57{\cdot}10^{-2}$	$4.28{\cdot}10^{-6}$	$1.73{\cdot}10^{-2}$	$2.17{\cdot}10^{-6}$	$7.35{\cdot}10^{-5}$	$5.32{\cdot}10^{-9}$	3.75	$1.40{\cdot}10^{-4}$
Failed	85.84	$2.60{\cdot}10^{-2}$	$1.00{\cdot}10^{-5}$	$2.21{\cdot}10^{-2}$	$4.93{\cdot}10^{-6}$	$1.10{\cdot}10^{-4}$	$1.22{\cdot}10^{-8}$	2.41	$3.33{\cdot}10^{-4}$
Successful	2.71	$2.55{\cdot}10^{-2}$	$1.17{\cdot}10^{-7}$	$1.39{\cdot}10^{-2}$	$1.82{\cdot}10^{-7}$	$4.74{\cdot}10^{-5}$	$3.56{\cdot}10^{-10}$	4.72	0

requested by tasks at submission time and the task priority. In the Job event table, there is one specific feature, termed "final event" showing the final outcome of jobs, namely successful or failed. All events are recorded in a microsecond granularity. All resources are given in a normalized value ranging between $[0, 1]$, against the maximum amount of resources available on machines.

B. Data Filtering

We face several challenges in gathering the attributes previously outlined. All the trace tables have different logging granularities, structures, and even missing information. Our analysis spans over very large volume of tables, e.g., 10.78 GB for the Task events table. Collecting statistics that require attributes from multiple tables leads to complex queries and results into a high computational time.

To better sanitize the data, we filter jobs that (i) have no active tasks, (ii) have some missing information in their tasks records (for example, requested resources for some tasks are missing; final job or task outcomes are not provided; etc.), (iii) are in execution at the beginning or end of the trace, and (iv) fail before having been scheduled on machines. Overall, our filtering process removes 7732 jobs out of 672074 (1.15%) from the data set.

C. Basic Statistics of Failed vs. Successful Jobs

After the aforementioned filtering process, we classify jobs into two categories: failed and successful jobs, based on their final event recorded in the Job events table. Here, we give an overall view on the similarity between the two types of jobs, using static job characteristics that are known upon job arrivals, i.e., number of tasks, requested resources, and priorities. As requested resources and priorities are provided by users and specified at task level, each job has multiple such values depending on its number of tasks. We thus propose to characterize each job with the average and standard deviation of requested resources and priorities, computed over all its tasks. The standard deviation values explain the disparity among tasks.

Table II summarizes the mean values of aforementioned statistics for failed and successful jobs. One can see that on average failed jobs have higher numbers of tasks and lower priorities, compared to successful jobs. This observation indicates the importance of job size in terms of number of tasks and priorities in explaining failed jobs. In terms of average requested resources, failed jobs request similar amount of CPU, higher RAM, and higher DISK than successful jobs. As for the standard deviation across tasks' requested resources, successful jobs always show lower values than failed ones, meaning that the disparity among successful jobs' tasks is lower.

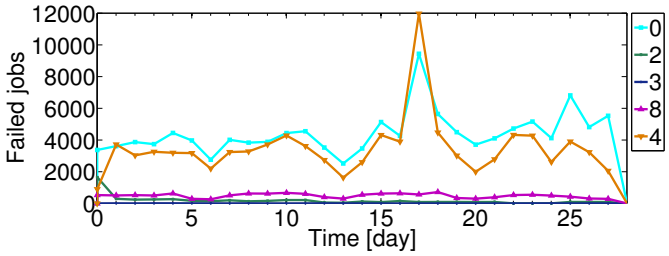


Fig. 2. Number of failed jobs per priority per day, selected from priority 0, 2, 3, 4, and 8.

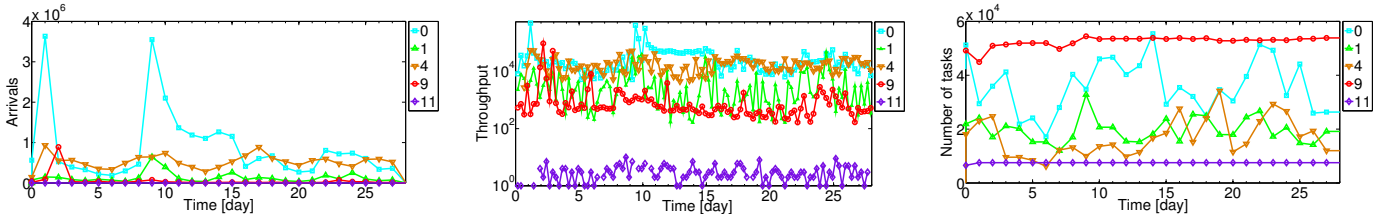
III. TIME-VARYING FAILED JOBS AND SYSTEM LOADS

Prior to developing the prediction model and the mitigation policy, we study the time-varying behavior of failed jobs and investigate their dependency on system loads via eye ball solution. Motivated by the importance of priority in these traces, all analysis here consider also priorities. To such an end, we first collect the number of failed jobs per priority per day and summarize the result in Figure 2. In the figure, we only show those priorities with a significant number of jobs. Note that the trace only provides priority for tasks, so we obtain priorities for jobs by first averaging their tasks priorities and then rounding off. In this way, we also obtain for job priorities the same of range of task priorities, i.e., from 0 to 11.

One can clearly observe from Figure 2 that there are strong trends of time variability in failed jobs, and such trends vary across priorities. Jobs of priority 0 and priority 4 have the highest number of failures. Also, on day 17^{th}, there is a surge of failed jobs for both priority 0 and priority 4. As for the other priorities, they experience similar levels of failed jobs and also a milder pattern of time variability. We thus draw the conclusion that priority is a crucial factor in failed jobs, but the ranking of priority does not play a role. Moreover, the strong time variability suggests that including time-relevant information, such as the most recent system loads, can explain failed jobs better.

A. Dependency on System Load

Prior work on system reliability [12] points out that there exist positive correlations between application performance and system load. To demystify the time-varying behavior of



(a) Daily arrivals. (b) Hourly throughput. (c) Daily number of tasks.

Fig. 3. Trends of three system load indicators for priority 0, 1, 4, 9, and 11.

failed jobs across priorities, we also resort to the system load. In particular, we propose to use task level statistics as load indicators, instead of jobs, due to disparate task fanout levels. We select three types of metrics, namely *arrival*, *throughput*, and *number of tasks*, as indicators for load increment, decrement, and instantaneous state, respectively. The throughput per priority is defined as the number of tasks leaving the system, regardless of their outcome being success or failure. We sample these metrics with 5 minutes granularity. For a better readability and comparison with jobs in Figure 2, we mainly present daily results in Figure 3, except for the throughput that is shown in hourly window. Moreover, we use a logarithmic scale for the throughput, due to the high disparity of trends between different priorities. We only include the five priorities that have higher values and variability, such that one can better observe the system dynamics in different granularities. Overall, all three metrics show time-variability, especially for priority 0 and priority 4.

One can see that throughput trends grossly follow arrival trends across priorities, even though two different sampling granularities are applied. The differences between arrivals and throughput result into fluctuations in the number of tasks as shown by Figure 3 (c), especially for priority 0 and priority 4. As these two priorities have a dominant appearance in failed jobs, we focus on investigating the dependency between their failed jobs and load indicators. First of all, visual inspection on the spikes shows that peaks of failed jobs in Figure 2 do not necessarily coincide with peaks of arrivals and throughput. For priority 0, arrival peaks happen on day 1 and 9, whereas the peak of failed jobs is on day 17. To our surprise, peaks of number of tasks in the system don't really coincide with failed jobs. One possible reason could be that the presented granularity is too gross to spot the transient correlation between failed jobs and the number of tasks. We formally address such a shortcoming by applying various machine learning techniques on 5 minutes data in the following section.

IV. Proposed Methodology

In this section, we propose an on-line prediction model that can predict expected job outcomes, i.e., successful vs. failed, upon job arrivals and a delay-based mitigation policy that proactively terminates predicted-to-fail jobs with the final goal of minimizing resource waste. The main objectives of the proposed methodology are twofold: (1) to accurately predict job failures in highly heterogenous and time-varying big-data systems, and (2) to reduce false terminations of jobs that could have been completed successfully.

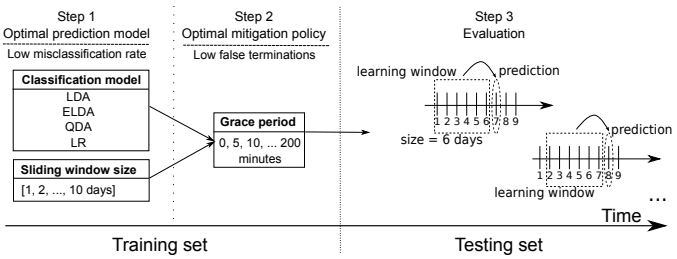


Fig. 4. Summary of the proposed methodology and evaluation.

To such an end, we apply a number of well-known machine learning techniques, in particular supervised classification techniques, to develop a job prediction model in a sliding window fashion. As such, we can well capture the time-varying patterns of job failures and their dependency on system load. The input of the model is a set of features, which describes key job attributes and system attributes collected upon job arrivals. Moreover, based on the prediction results, we propose a mitigation policy that proactively terminates jobs which are predicted to fail after a certain grace period. The duration of the grace period is set in order to strike a tradeoff between resource waste resulted from job failures and false terminations of successful jobs.

Following conventional statistical learning approaches, we partition the traces data into a training set and a testing set. We first search for the optimal choice of the prediction model and mitigation policy on the training set during the training phase. The selection criteria are the low misclassification rate as well as the low number of false terminations on successful jobs. Afterward, we apply the prediction model and mitigation policy that result from the training phase on the testing set. Figure 4 shows the skeleton of the proposed solution.

In the following, we first define the quantitative metrics of interest, and introduce the feature sets. Then, we provide an overview on the on-line prediction model and detailed design of the mitigation policy.

A. Quantitative Metrics of Interest

We consider three main quantitative metrics: misclassification rate, false positive rate, and false negative rate. The *misclassification rate* is defined as the number of misclassifications divided by the total number of jobs. A *false positive*

TABLE III
LIST OF ALL STATIC AND SYSTEM FEATURES CONSIDERED FOR EVERY JOB ARRIVAL.

Static	Number of tasks AVG/STD task requested CPU AVG/STD task requested DISK	AVG/STD task priority AVG/STD task requested RAM
System	AVG/STD lower priority arrivals AVG/STD same priority arrivals AVG/STD higher priority arrivals AVG/STD lower priority throughput AVG/STD same priority throughput AVG/STD higher priority throughput AVG/STD lower priority number of tasks AVG/STD same priority number of tasks AVG/STD higher priority number of tasks	AVG/STD lower priority variation of arrivals AVG/STD same priority variation of arrivals AVG/STD higher priority variation of arrivals AVG/STD lower priority variation of throughput AVG/STD same priority variation of throughput AVG/STD higher priority variation of throughput AVG/STD lower priority variation of number of tasks AVG/STD same priority variation of number of tasks AVG/STD higher priority variation of number of tasks

happens when a job is classified as successful, though its real outcome is a failure. On the contrary, a *false negative* happens when a successful job is wrongly classified as failed job by the model. Following these definitions, we define the *false positive rate* as the number of false positives divided by the total number of jobs. Similarly, the *false negative rate* is the ratio between the number of false negatives and the number of jobs considered. The misclassification rate can be also defined as the sum of the false positive rate and the false negative rate. It is worth noting that a high false negative rate has a strong negative impact on the users' and clients' satisfaction. Indeed, while allowing the execution of failed jobs impacts on system resources, denying execution to a successful job can greatly impair users and customers' satisfaction at datacenters. For this reason, we focus more on reducing the false negative rate than the false positive rate in our methodology.

B. Features

As our goal is to accurately classify jobs into two distinct classes, i.e., successful or failed, upon job arrivals, we leverage features that can be obtained with low overhead and capture instantaneous system dynamics. In particular, we consider two types of features: *static* features, i.e., related to static information about jobs, and *system* features, i.e., related to the system load.

1) Static features: Jobs are composed of several tasks, each of which has a given priority and requests a certain amount of resources upon arrival, i.e., CPU, RAM, and DISK. This information is provided by users upon submission of their jobs. Thus, we define static features that describe the dissimilarity between tasks of the same job: the number of tasks within a job, the task priority, and task requested resources. We consider all three types of resources, namely, CPU, RAM, and DISK. Moreover, due to the multi-task nature of big data jobs, we use the average and the standard deviation of task priority and requested CPU, RAM, and DISK. As s result, we have a total of 9 static features for every job.

2) System features: Motivated by the gross correlation between the three metrics of system load and job failures described in Section III, we consider the number of tasks, arrivals, and throughput as key types of system features. In contrast to static features, system features need to be collected on-line upon job arrivals. To compute those values, we employ a minimum sampling window, i.e., 5 minutes. We compute the number of arrivals and throughput for each window and monitor the number of tasks at the beginning of the window. When jobs arrive to the system, we check system features in the most recent window. Note that the window length should be short enough to capture the transient load changes, but also long enough to avoid a high monitoring overhead. As the average execution time per job in the trace considered is rather long, i.e., 46 minutes, a 5 minute sampling time is a good choice here.

In addition to three system load indicators, the dimensions of system features are further expanded by considering priorities and transient changes in the recent past. To take into account the influence of different priorities on jobs, we stress the loads for higher priority, lower priority, and the same priority for incoming jobs. Consequently, for each job, we compute the aforementioned load indicators for three categories of priority: (i) its priority (ii) lower priority and (iii) higher priority. For example, when a job with priority 7 arrives to the system, we compute three different values of task arrivals, considering tasks of priority 7, tasks with priority among $[0, 6]$, and tasks with priority among $[8, 11]$, respectively. A similar computation needs to be applied for the throughput, as well as for the number of tasks.

Finally, to factor in the stability of the system state, we include metrics that explain the difference between the most two recent sampling windows. Essentially, we consider the aforementioned features in the most recent window, as well as their variation between the most recent window and the preceding window. Taking as example a job of priority 7 again, we include the difference of arrivals between the most two recent windows for all three categories of priorities. This also applies on the throughput and the number of tasks. In summary, we compute 36 different system features, resulting from a combination of three load indicators, three priority categories, two sampling windows, and two different statistics (average and standard deviation) upon job arrivals. We summarize all the considered features in Table III.

C. On-line Prediction Model

Due to the time-varying nature of jobs and tasks, and the high dynamicity of the system [8], we propose to develop an on-line prediction model in a sliding window fashion.

The objective of such a procedure is to recognize changes in workloads or system, and adapt the prediction model to such changes. The proposed procedure evolves as follows: we compute a new classification model at the end of each day, and we use it to predict the outcomes of job arrivals within the following 24 hours. The classification model for each day is based on historical information of the learning window. In particular, we train the classification model for day d_i by using jobs terminations dated from day d_{i-N} to d_{i-1}, where N is the length of the window.

In terms of prediction models, we consider four types of supervised classification techniques that classify job outcomes. All the considered techniques learn a function separating jobs of the feature set in two different classes, i.e., successful and failed. Jobs are considered as points in a multi-dimensional space, and the function learns a hyperplane separating this space in two half spaces, one per class. The number of dimensions is equal to the number of different features considered for each job. During the training phase, all techniques compute this function to minimize the misclassification error on the training set.

The particular classification models considered are:

1) Linear Discriminant Analysis (LDA).
2) Linear Discriminant Analysis on expanded basis (ELDA).
3) Quadratic Discriminant Analysis (QDA).
4) Logistic Regression (LR).

The difference between the classification techniques above lies in the order of the function learnt and the distribution of the feature set. In particular, LDA computes a linear hyperplane separating jobs into the two classes: denoting the training set as a vector $\mathbf{x}$, the linear hyperplane has the form $\mathbf{a}^T\mathbf{x} + b$, where $\mathbf{a}$ is a vector of weights and b is a scalar that are learned from the training set. The classification works as follow: jobs belonging to one side of the hyperplane, i.e., $\mathbf{a}^T\mathbf{x} + b > 0$, are classified as successful, while the others are classified as failed.

ELDA is a variation of LDA, where a linear function is computed from a larger feature set. This *extended* set is derived from the original one, where we also consider the squared value of each feature and their product. For example, the extended set of an hypothetical feature set containing only two features, f and g, is composed by f, g, $f \cdot g$, f^2 and g^2. Generalizing this concept and applying it to our model, we have a total of 54 extended static features and 702 extended system features in ELDA, whereas LDA, QDA and LR only use a total of 45 features.

QDA is a generalization of LDA, where the separating hyperplane is a quadratic function on $\mathbf{x}$, i.e., $\mathbf{x}^T\mathbf{A}\mathbf{x}+\mathbf{b}^T\mathbf{x}+c$, where $\mathbf{A}$, $\mathbf{b}$ and c are a diagonal matrix, a vector and a scalar of weights, respectively, learned from the feature set. A quadratic model usually classifies data better than a first-order one, at the cost of more weights to estimate.

LDA, ELDA and QDA all assume that features are independent and normally distributed. On the contrary, the fourth method considered, i.e., LR, doesn't have this requirement, being much more flexible, and it still computes a linear hyperplane. As a drawback, LR requires a much larger training set in order to achieve good classification results. As these techniques are well-known, we refer readers to [13] for detailed explanation and derivations. Prior to computing the classification models, we normalize all values of the feature set.

As for the choice of the classification model and the size of the on-line learning window, we resort to an extensive evaluation on the training data set. We essentially experiment different combinations of classification models and sizes of the learning window. The optimal choice is the combination which results into the lowest misclassification rate.

D. Mitigation Policy

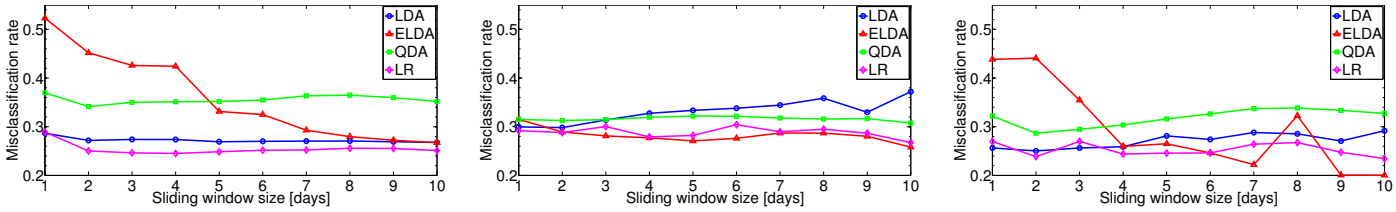
To minimize the resource waste resulting from failed jobs, we develop a delay-based mitigation policy, which proactively terminates predicted-to-fail jobs after a certain delay from job's scheduling time. The objective of such waiting delay is to reduce the number of false negatives that cause great dissatisfaction from datacenter users and clients. Nevertheless, the reduction of resource waste achieved by terminating predicted-to-succeed jobs decreases when the waiting delay increases. Essentially, the prediction model first achieves low misclassification rate, thereafter the mitigation policy minimizes the impact of false negatives.

The proposed mitigation policy applies on every job arrival, using two key inputs: the prediction results of every job arrival and the delay duration, termed *grace period*. When jobs are predicted to fail upon their arrival, we first grant their execution, then we proactively terminate them after the grace period, if jobs are still running. As for the predicted-to-succeed jobs, we do not interfere with their executions. A larger grace period causes a lower number of false negatives at the expense of higher resource waste, while a smaller grace period inevitably increases the number of false negatives, but leads to a lower resource waste.

As for the value of the grace period, we propose to determine the optimal one in the training phase from a set of pre-defined values. Essentially, during the training phase, we not only learn the model, but also simultaneously search for optimal window size as well as the optimal length of grace period.

V. Evaluation

In this section, we present the evaluation of the proposed methodology on determining the optimal job prediction model and mitigation policy as well as their efficacy in reducing resources wasted by job failures. Our evaluation is based on 26 days of the traces, which we divide into two distinct sets for the training and testing phases. The first 14 days of the traces compose the training data, while the remainder of the traces composes the testing data. As described in Section IV, the aim of the training phase is to select the best classification technique for job arrivals, the optimal parameter for the on-line prediction model, i.e., the size of the learning window, and the



(a) Static only. (b) System only. (c) Static and system.

Fig. 5. Misclassification rates of feature sets: combinations of different classification models and learning window sizes.

optimal grace period for our delay-based mitigation policy. In the testing phase, we focus on reporting the misclassification rate, the false terminations of predicted-to-succeed jobs, and the reduction of resource waste.

We structure the evaluation into three parts: (1) deriving an optimal on-line prediction model for job outcomes, (2) determining an optimal delay-based mitigation policy, and (3) testing the proposed model and policy. The first two parts are conducted on the training data, whereas the last part is conducted on the testing data. To develop an optimal on-line prediction model, we evaluate different combinations of classification models, namely, LDA, ELDA, QDA, and LR, and sizes of the training window, ranging from one to ten days. As for the mitigation policy, we simulate different lengths of the grace period prior to terminating predicted-to-fail jobs, in combination with the optimal on-line model identified in the first part.

A. Optimal On-line Prediction Model

Here, we derive the optimal on-line prediction model by evaluating combinations of classification models and learning window sizes. The classification models considered are LDA, ELDA, QDA, and LR (see Section IV-C for details), while the learning window size ranges from one to ten days. The evaluation criterion is the misclassification rate. Moreover, to highlight the effectiveness of the proposed features, we also evaluate different combinations of feature sets, i.e., static feature only, system feature only, and both static and system features. The results are summarized in Figure 5.

Static Features Only: When training classification models with only static features, one can see from Figure 5 (a) that LR has the lowest misclassification rate across all window sizes, with values as low as 0.25. When the window size is 4 days, LR achieves the lowest misclassification rate. LDA achieves not only the second lowest misclassification rate but also good stability in different learning window sizes. Interestingly, the misclassification rate of applying ELDA drastically reduces with increasing window sizes. Indeed, ELDA and LDA have similar misclassification rates when the window size is equal to ten days. The worst classification model when using only static features is given by QDA, with misclassification rate greater than 0.34 on all window sizes.

System Features Only: The behaviors of all model combinations here are different from the patterns in the previous case, as shown in Figure 5 (b). First of all, LDA has the worst prediction result for most sizes of the learning window, and its misclassification rate increases with increasing learning window size, reaching values as high as 0.37. As for QDA, the misclassification rate is slightly improved from the case of static features and is very constant across window sizes, with values around 0.32. ELDA and LR are the two techniques that yield best classification results, with an average misclassification rate around 0.28. Both their misclassification rates show very similar trends, e.g., the trends slowly decay with increasing window sizes. Particularly, when the learning window size is very small, i.e., 1 or 2 days, LR outperforms ELDA. Nevertheless, when classification techniques use only system features to build classification models, the misclassification is rather high. Indeed, no model has a misclassification rate as low as 0.25, i.e., the best result that we can achieve by using static features.

Static and System Features: From Figure 5 (c), one can see that all four classification models can reach better results, in comparison with the cases of using static or system features only. This observation confirms the effectiveness of using both feature sets. Certain combinations of models and window sizes can result in misclassifications rate lower than 0.25. Particularly, ELDA is able to predict job outcomes with a misclassification rate as low as 0.2 when the model is trained on nine or more days. Another worth noting observation is that LDA, QDA and LR mainly show only mild variations across different window sizes for all three feature sets, whereas ELDA can greatly benefit from increasing window sizes when using static features or both features. Combing all observations in the training phase, we thus conclude that ELDA with a large learning window, i.e., 10 days, can best predict outcomes upon job arrival on the training set.

B. Optimal Mitigation Policy

In this section, we determine the optimal length of the grace period used in the mitigation policy that proactively terminates predicted-to-fail jobs. As mentioned in Section IV, a good prediction model has a low error rate, but can still originate several false negatives that can have a tremendous impact on datacenter users. Allowing execution of jobs that are predicted to fail for some time, i.e., the grace period, can significantly lower false job terminations, while still reducing resource waste by a high amount. To differentiate the original false negatives obtained from the prediction model, we term the number of false terminations resulting from our proposed

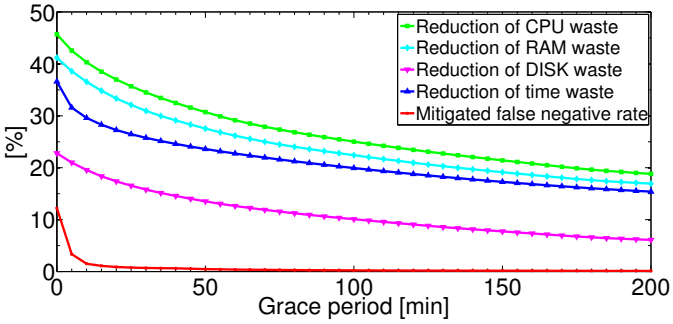


Fig. 6. Mitigated false negative rate and reduction of resource waste applying different grace periods.

delayed-based mitigation policy as *mitigated false negatives*.

To evaluate the optimal grace period in our mitigation policy, we employ a numerical simulation that computes the reduction of resource waste and mitigated false negatives with several values of grace period. The prediction outcomes used in our mitigation policy are provided by ELDA with a sliding window size of 10 days. In particular, the values considered for the grace period range from 0 (the job is terminated immediately) to 200 minutes, increasing by 5 minutes. To quantify the amount of resources gained by terminating jobs executions, we use the metric of *reduction of resource waste* (RRW), defined as $RRW = 1 - \frac{R_s}{R_o}$, where R_o denotes the amount of resources originally consumed by all jobs in the trace, and R_s refers to the amount of resources consumed by all jobs when applying our mitigation policy. More precisely, the amount of resources consumed by a job is the product of its number of tasks, the average resources requested by its tasks, and its execution time. Note that, under the proposed mitigation policy, execution times of failed jobs are lower than or equal to the grace period. R_s and R_o are then obtained by summing up the resources consumed by all jobs. We compute four different values of RRW, considering machine time, CPU, RAM and DISK. As for the mitigated false negatives, i.e., incorrect terminations of successful jobs that are predicted to fail, we use the *mitigated false negative rate*, defined as the number of mitigated false negatives, divided by the number of jobs considered by the simulation. We summarize both RRW and the mitigated false negative rate in Figure 6.

From Figure 6, we can see how our mitigation policy is able to reduce the resource waste considerably, even with a large grace period. Among four types of resources, CPU is the one that benefits mostly from the mitigation policy, with a reduction of resource waste ranging from about 46% to 19% in our simulation. Volatile memory closely follows the CPU trend, even if the actual reduction of wasted RAM is lower than the CPU one. On the contrary, storage is the resource that is less affected by our mitigation policy, as terminated jobs request small amounts of DISK. Nevertheless, we are able to save up to 23% of mass storage by immediately terminating jobs that are predicted to fail. Regarding the computational time, our policy is able to save 36% of machine time by terminating jobs as soon as they arrive to the system. This value decreases sharply with increasing grace periods, as the majority of failed jobs has short execution time.

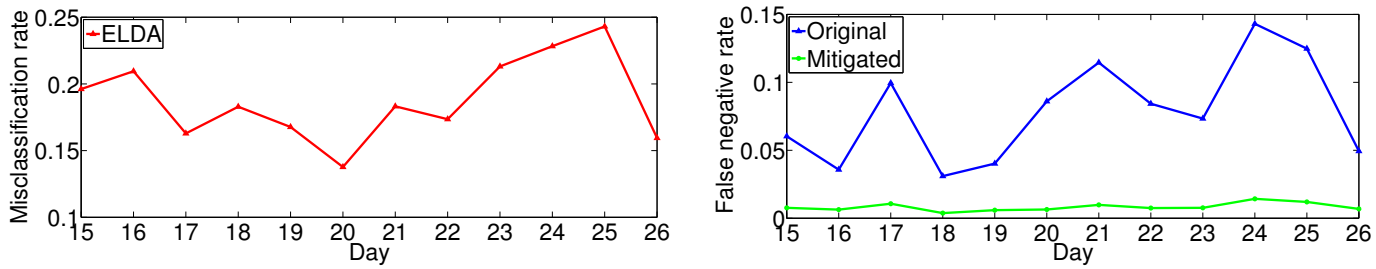
In terms of mitigated false negatives, from the initial value of 12.3%, this rate decreases swiftly below 1% when the grace period is larger than 20 minutes. This indicates that a small grace period can guarantee a low mitigated false negative rate, while still allowing a substantial reduction of resource waste. Starting from this observation, we propose to choose a grace period that maximizes the reduction of resource waste, while keeping the mitigated false negative rate lower than a certain threshold, i.e., 1%. As such, the optimal length of the grace period is equal to 20 minutes, which results into a mitigated false negative rate of 0.88% and reduction of resource waste equal to 37.0%, 33.4%, 17.4% and 27.3% for CPU, RAM, DISK and computational time, respectively.

In conclusion, we extensively evaluate all combinations of classification models, feature sets, sliding window sizes and grace periods in the training phase. The optimal prediction model is to apply ELDA with a sliding window of 10 days, using both static and system features. To achieve 1% of mitigated false negative rate, the grace period of the mitigation policy should be 20 minutes, meaning that we only terminate predicted-to-fail job executions after 20 minutes from their scheduling. In the following subsection, we apply this particular model and policy on the testing set.

C. Testing Phase

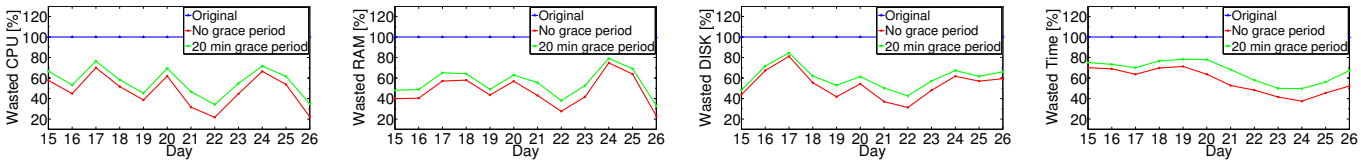
To evaluate the robustness of the proposed prediction methodology to dynamical changes in the workload or system properties, we test the optimal model and policy on the second partition of the traces. Differently from the previous subsection, we also focus on presenting daily results. Particularly, we first show the daily misclassification rate and false negative rates in Figure 7. Secondly, we present the resource waste under the following scenarios: 1) original trace (no mitigation policy applied), 2) mitigation policy with no grace period, and 3) mitigation policy with grace period of 20 minutes. Figure 8 summarizes their comparison for computational time, CPU, RAM and DISK.

The daily misclassification rate shown in Figure 7 (a) is promising, i.e., the misclassification rate in most days is lower than 0.2, the best result of the training phase. Another worth noting observation is that the misclassification rate is rather constant, even though the original failed jobs show a strong time variability, as presented in Figure 2. This implies that the proposed on-line prediction model employs crucial features and thus well capture the time-variability of job failures. As for false negative rates in Figure 7 (b), we present two values: the original false negative rate based on ELDA, and the mitigated false negative rate after applying the optimal delayed-based mitigation policy. Clearly, the mitigated false negative rate is significantly lower than the one directly resulting from ELDA. Moreover, the mitigated false negative rate is very stable, i.e., below 1%. Such an observation again confirms



(a) Daily ELDA misclassification rate. (b) Daily false negative rates.

Fig. 7. Results of applying Linear Discriminant Analysis on expanded basis (ELDA) with a sliding window of 10 days and a mitigation policy with 20 minutes grace period.



(a) CPU. (b) RAM. (c) DISK. (d) Computational time.

Fig. 8. Comparison of daily resource waste: no mitigation policy, mitigation policy with no grace period, and mitigation policy with 20 minutes grace period.

the effectiveness of terminating predicted-to-fail jobs after the grace period.

Regarding the resource waste, we present in Figure 8 the percentage of resource wasted by our mitigation policy, normalized by the resource waste of the original traces. Additionally, we plot a straight line of 100% that refers to the original resource waste in the traces. One can immediately observe how a mitigation policy that proactively terminates predicted-to-fail jobs drastically reduces resource waste, in terms of machine times, CPU, RAM, and DISK. As expected, percentages of resource waste decrease slightly from using no grace period to using 20 minutes grace period in the mitigation policy. These observations, combined with the great improvement of mitigated false negative rate in Figure 7, confirm that the grace period strikes a good tradeoff between resource waste and false terminations of successful jobs. Furthermore, we zoom into the comparison among the trends of four resources. On the one hand, CPU and RAM show a strong similarity in their pattern of resource wastes, i.e., they increase and decrease on similar days. On the other hand, machine time exhibits a very different pattern. For example, on day 24^{th}, the machine time has a dip, while the other three resources have peaks.

Finally, in terms of average over the entire testing set, the reduction of resource waste is equal to 46.7%, 47.1%, 40.5%, and 33.2% for CPU, RAM, DISK and computational time, respectively, with a mitigated false negative rate of 1.05%. These results confirm that our predictive model, combined with the proposed mitigation policy, is able to effectively reduce resource waste in big datacenters, where the workload is highly dynamic and diversified and system load changes frequently.

VI. Related Work

Improving application and system reliability is a fundamental operation in computational data centers, whose architectures and running applications have become increasingly complex. Although there is a rich number of field studies on hardware failures [14]–[16], application job failures are often overlooked. We summarize prior works related to our paper in three distinct categories: (1) existing analysis on the same Google traces, (2) studies on failure modeling and prediction, and (3) prior work on improving system dependability.

Google Traces Analysis

These traces [3] have been analyzed by several prior works, including workload characterization [8], [9] and classification of tasks via statistical techniques [2], [17]. The first and most complete analysis is the one conducted by Reiss et al. [8] that highlights the heterogeneous and highly dynamic nature of big data workloads. However, this work does not provide a comprehensive analysis on job failures and system load, limiting its focus to the general behavior of the system. The concept of wasted resources is introduced by Liu et al. [9] with a characterization study on wasted CPU cycles on a single day of the traces, while we propose a method to reduce the resource waste thanks to an on-line prediction model and a subsequent delay-based mitigation policy, providing an extensive evaluation over all days of the traces. Di et al. [2], [17] apply unsupervised learning techniques like k-mean to build clusters of jobs, and present a small discussion on task failures on a single machine. Overall, existing literature on these Google traces does overlook failed jobs executions in big data clusters, and doesn't provide neither prediction methods on job failures nor policies to mitigate the resource waste.

Failure Modeling and Prediction

Several related studies aim to analyze failure patterns in distributed systems [14], [18], high-performance supercomputers [16], [19] and storage components [20], [21]. Failure

modeling is a fundamental contribution towards failure prediction and reliability improvement. Potharaju et al. show that a two-component lognormal distribution can effectively model failures occurring at middleboxes [22] and network components [14], while Ford et al. [23] use Markov chains to model storage availability in distributed systems. Liang et al. [16] derive three different prediction algorithms based on a field study on hardware failures in the BlueGene/L supercomputer. Moreover, reliability has long been a major concern in the data storage area, such as Redundant Array of Inexpensive Disks (RAID), where queueing based modeling techniques have been extensively applied to predict reliability [20], [21]. While the focus of prior modeling work centers on static systems, we provide an on-line classification model that is able to predict job outcomes in presence of dynamic workload and unstable system behavior.

Improving Dependability

Recognizing the prevalence of failures happening at all layers of systems, e.g., hardware and software, designing fault tolerance systems and failure recovery strategies has long been an important issue, especially for distributed systems [24]. A typical practice to increase the dependability of different system layers and components is to add redundancy. Barroso et al. [1] replicate job executions to increase reliability as well as application performance. The basic principle of RAIDs is to ensure data availability by increasing the replication factor of data [25]. Pullu [26] summarizes practical techniques to increase software determinability of high performance systems. Tucek et al. [27], [28] develop novel techniques to ensure accurate execution of software and defend it against software worms. Differently from the prior work tackling the root causes of failures, we focus on developing a greedy approach that fast mitigates the resource waste of job failures.

VII. Conclusion

Motivated by the significant amount of resource waste, in terms of computational time, CPU, RAM and DISK, caused by job failures at big data clusters, we aim to capture failed jobs upon their arrival and minimize the resulting resource waste. To incorporate transient and complex system dynamics, we consider extensive static and system features that capture the disparities of jobs' multiple tasks and system load across priorities. We first explore four supervised classification techniques, namely LDA, ELDA, QDA, and LR, in an sliding window fashion, when developing an on-line prediction model for job failures. Based on the prediction results, we develop a delay-based mitigation policy that proactively terminates predicted-to-fail jobs after a certain grace period. The optimal choice of the classification technique, size of the sliding window, and length of grace period are determined during the training phase, so to achieve low misclassification rate and mitigated false negative rate. Our evaluation on the testing set shows that we falsely terminate only 1% of successful jobs, and, most importantly, we reduce resource waste by 33.2%, 46.7%, 47.1%, and 40.5% for machine time, CPU, RAM, and DISK, respectively, in comparison to the original resource waste.

References

[1] L. Barroso, J. Dean, and U. Hölzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, Mar. 2003.

[2] S. Di, D. Kondo, and W. Cirne, "Characterization and Comparison of Cloud versus Grid Workloads," in *IEEE CLUSTER*, 2012, pp. 230–238.

[3] J. Wilkes, "More Google cluster data," Google research blog. https://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1, Nov 2011.

[4] B. Schroeder and G. A. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," *IEEE Trans. Dependable Sec. Comput.*, vol. 7, no. 4, pp. 337–351, 2010.

[5] ——, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" in *USENIX FAST*, 2007, pp. 1–16.

[6] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes — A Comprehensive Study on Real World Concurrency Bug Characteristics," in *ASPLOS*, 2008, pp. 329–339.

[7] D. Yuan, D. Park, P. Huang, Y. Liu, M. Lee, Y. Zhou, and S. Savage, "Be Conservative: Enhancing Failure Diagnosis with Proactive Logging," in *USENIX OSDI*, 2012, pp. 293–306.

[8] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *ACM SoCC*, 2012, pp. 7:1–7:13.

[9] Z. Liu and S. Cho, "Characterizing Machines and Workloads on a Google Cluster," in *SRMPDS*, 2012, pp. 397–403.

[10] B. Cho, M. Rahman, T. Chajed, I. Gupta, C. Abad, N. Roberts, and P. Lin, "Natjam: Design and Evaluation of Eviction Policies for Supporting Priorities and Deadlines in Mapreduce Clusters," in *ACM SoCC*, 2013, pp. 6:1–6:17.

[11] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Technical Report, revised 2012.03.20. http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

[12] R. Birke, I. Giurgiu, L. Chen, D. Wiessmann, and T. Engbersen, "Failures Analysis of Virtual and Physical Machines: Patterns, Causes and Characteristics," in *IEEE/IFIP DSN*, 2014, pp. 1–12.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[14] R. Potharaju and N. Jain, "When the Network Crumbles: An Empirical Study of Cloud Network Failures and Their Impact on Services," in *ACM SoCC*, 2013, pp. 15:1–15:17.

[15] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM Errors in the Wild: A Large-scale Field Study," in *ACM SIGMETRICS*, 2009, pp. 193–204.

[16] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, "BlueGene/L Failure Analysis and Prediction Models," in *IEEE/IFIP DSN*, 2006, pp. 425–434.

[17] S. Di, D. Kondo, and C. Franck, "Characterizing Cloud Applications on a Google Data Center," in *ICPP*, 2013, pp. 468–473.

[18] K. V. Vishwanath and N. Nagappan, "Characterizing Cloud Computing Hardware Reliability," in *ACM SoCC*, 2010, pp. 193–204.

[19] K. Trivedi, *Probability & Statistics With Reliability, Queuing And Computer Science Applications*, 2nd ed. John Wiley & Sons, 2008.

[20] I. Iliadis, "Reliability modeling of RAID storage systems with latent errors," in *IEEE/ACM MASCOTS*, 2009, pp. 1–12.

[21] V. Venkatesan and I. Iliadis, "Effect of latent errors on the reliability of data storage systems," in *IEEE/ACM MASCOTS*, 2013, pp. 293–297.

[22] R. Potharaju and N. Jain, "Demystifying the Dark Side of the Middle: A Field Study of Middlebox Failures in Datacenters," in *ACM IMC*, 2013, pp. 9–22.

[23] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in Globally Distributed Storage Systems," in *USENIX OSDI*, 2010, pp. 61–74.

[24] A. S. Tanenbaum and M. V. Steen, *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2006.

[25] I. Iliadis, R. Haas, X. Hu, and E. Eleftheriou, "Disk scrubbing versus intra-disk redundancy for high-reliability raid storage systems," in *ACM SIGMETRICS*, 2008, pp. 241–252.

[26] L. L. Pullu, *Software Fault Tolerance Techniques and Implementation*. Artech House Computing Library, 2001.

[27] J. Tucek, J. Newsome, S. Lu, C. Huang, S. Xanthos, D. Brumley, Y. Zhou, and D. X. Song, "Sweeper: a lightweight end-to-end system for defending against fast worms," in *EuroSys*, 2007, pp. 115–128.

[28] J. Tucek, W. Xiong, and Y. Zhou, "Efficient online validation with delta execution," in *ASPLOS*, 2009, pp. 193–204.