Catching Failures of Failures at Big-Data Clusters: a Two-Level Neural Network Approach

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*—Big-data applications are becoming the core of today's business operations, featuring complex data structures and high task fan-out. According to the publicly available Google trace, more than 40% of big-data jobs do not reach successful completion. Interestingly, a significant portion of tasks of such failed jobs undergo multiple types of repetitive failed executions and consume a non-negligible amount of resources. To conserve resources for big-data clusters, it is imperative to capture such failed tasks of failed jobs, a very challenging problem due to multiple types of failures associated with tasks and highly uneven tasks distribution. In this paper, we develop an on-line two-level Neural Network (NN) model which can accurately untangle the complex dependencies among tasks and jobs, and predict their execution classes in an extremely dynamic and heterogeneous system. Our proposed NN model predicts first the job class, and secondly three classes of failed tasks of failed jobs, based on a sliding learning window. Furthermore, we develop resource conservation policies that terminate failed tasks of failed jobs after a grace period that is derived from prediction confidences and task execution times. Overall, evaluating our results on a Google cluster trace, we are able to accurately capture failures of failures at big-data clusters, mitigate false negative tasks to 1%, and efficiently save system resources, achieving significant reductions of CPU, memory and disk consumption – as high as 49%.**

I. Introduction

Big-data applications present both opportunities and challenges for today's business operations. The high task fan-out of big-data jobs expedites data intensive processing, but at the same time exacerbates the complexity of managing such applications. As both workloads and systems show high degrees of complexity, it is no mean feat to optimize the application performance and the resource consumption efficiently. Moreover, according to the publicly available Google trace [1], more than 40% of jobs fail, consuming a non negligible amount of cluster resources. Among tasks of these failed jobs, 25% of them do not reach successful completion and leave the system in one of three failed execution types, namely eviction, fail, and kill. We summarize in Figure 1 the normalized machine time and resources consumed by these failed tasks of failed jobs, the so-called *failures of failures*. A daunting finding is that less than 25% of tasks actually consume the majority of resources in the cluster, with values around 65% for CPU, RAM and computational time, and 50% for DISK, respectively. Such a phenomenon is actually explained by the repetitive submissions of failed tasks. As such, failed tasks of failed jobs not only impair the application performance, but also greatly undermine the resource efficiency of the entire system. To improve the performance of both applications and systems in big-data clusters, it is imperative to capture and manage such failures of failures.

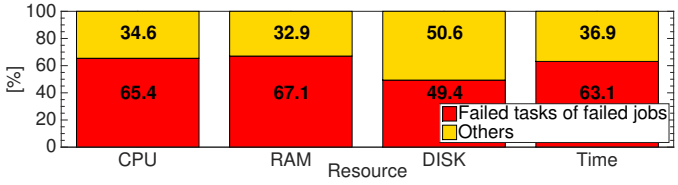


Fig. 1. Distribution of resource consumption of failed tasks of failed jobs and other types of tasks: CPU, RAM, DISK, and computational time.

The corresponding challenges arise from the following aspects. Failed tasks of failed jobs are a special kind of dependent failure pattern which is different from the focus of most of the prior art – single level failure patterns for applications or systems. Due to the multi-task nature of big-data applications, overlooking the dependency between tasks and jobs can easily lead to conflicting job classifications for tasks belonging to the same job. Last but not least, both workloads and systems are highly heterogenous and strongly time-varying, as jobs and tasks of different priorities request various amount of resources across time and the distribution of different classes of jobs/tasks is very unbalanced.

In this paper, we develop a novel methodology that can first predict dependent failure patterns of big-data jobs and tasks upon their arrival and secondly reduce the resource consumption of failed tasks of failed jobs by early termination. Motivated by the effectiveness of Neural Networks (NN) in capturing non-linear complex models, we develop a 2-level NN model which predicts first two job classes and then four tasks classes. To better capture the characteristics of application and system dynamics, the proposed 2-level NN model uses an on-line training methodology that considers a large amount of historical data in a moving window fashion. Moreover, we design a resource conservation policy that terminates three classes of failed tasks of failed jobs after a given *grace period*, derived from *prediction confidences* and tasks execution times. The key objective of the proposed policy is to minimize misclassifications of successful tasks – as terminating such tasks can impair user satisfaction – while conserving a high amount of cluster resources.

We use 29 days of workload collected from Google datacenters [1] as a case study. In particular, we extract several static and system features upon the arrival of jobs and tasks. Static features are specified by users upon job submission, e.g., task priority and the amount of requested resources for each task. System features capture the system dynamics across different task priorities, including arrival rates, throughput, and the number of tasks for each priority. Evaluation results show that our proposed 2-level NN model can achieve a good accuracy, particularly at the job level. Most importantly, our proposed resource conservation policy is able to mitigate false negatives to 1%, as well as obtain a significant reduction of resource consumption, i.e., more than 45% for CPU and RAM, and around 20% for DISK and computational time.

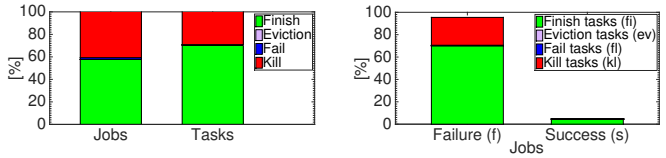
Our scientific contributions are threefold. First, we develop a novel 2-level NN model to capture the complex and dependent failure patterns in big-data applications. Secondly, by leveraging prediction confidences, we can effectively address the negative impact of false negative jobs and tasks. Finally, we provide a complete methodology for identifying, predicting, and mitigating failures of failures in large-scale big-data clusters. Our solution strikes a good balance between prediction accuracy, resource conservation, and unnecessary task terminations.

The remainder of the paper is organized as follows. Section II provides a basic description of the used data set. We detail our 2-level NN model and the design of the resource conservation policy in Section III. In Section IV, we present extensive evaluation results, followed by a discussion on related work in Section V. Section VI concludes the paper with a summary of our work.

II. Data and System description

Our analysis is based on a Google trace [1] which contains field data about rich heterogeneous workloads executing on a large heterogeneous cluster for 29 days. To the best of our knowledge, this is the only publicly available trace providing detailed information on job and task failures, along with their root causes. Here, we first present the definition of job and task types at Google datacenter; then, we summarize the basic statistics regarding different classes of jobs and tasks in this system. We refer the reader to our previous work [2] for a more detailed description on the trace and our data filtering process.

1) Trace: The trace contains rich information about workload submitted by users from the midnight of May 1^{st}, 2011 to the midnight of May 29^{th}, 2011. Users of the system submit *jobs* to the cluster, each of which containing multiple *tasks*, ranging from 1 to 90050. A task runs on a single machine at a time, and can experience multiple *events* before leaving the system permanently. Jobs, tasks, and events are labeled with the following four *types*: finish (i.e., successful completion), eviction (i.e., preemption due to higher priority tasks), fail (i.e., occurrence of a task internal error), and kill (i.e., arbitrary termination by the users or by the scheduler). The last three types are considered as failed, whereas the first type is regarded as successful. Types of jobs and events are given directly by the trace, whereas the types of tasks are given by the label of their last event. Note that we assign types to tasks only when they leave the system permanently. Upon job submission, users specify the resources required by their tasks, i.e., CPU, RAM, and DISK. CPU represents the maximum number of cores that a task can use, while RAM (DISK) specifies the maximum amount of volatile (storage) memory. All resources are given in a normalized value ranging between $[0, 1]$, against the maximum amount of resources available on machines. Furthermore, users assign a priority to each task, ranging from 0 to 11, where higher values characterize important tasks. Note that requested resources and priorities are defined at the task level upon job arrival.



(a) Distribution of jobs and tasks in the four types.

(b) Distribution of tasks in the eight classes.

Fig. 2. Job and task distributions.

2) Basic Statistics of Jobs and Tasks: We first summarize the distribution of jobs and tasks in Figure 2(a) which reports the percentage of each type without considering the dependency among jobs and tasks. For both jobs and tasks there are two dominant types, i.e., finish and kill. The distribution of job types leads us to focus on only two classes of jobs, i.e., success and failure, where the latter is composed of the three failed types. To factor in the dependency of tasks and jobs, we define 8 different task classes by the pair of job class and task type, (J, T). For each task i, its job class is either success (s) or failure (f), $J_i \in \{s, f\}$, and its task type can be finish (fi), eviction (ev), fail (fl), or kill (kl), $T_i \in \{fi, ev, fl, kl\}$. We depict the distribution of task classes in Figure 2(b). There are three dominant task classes, i.e., (s, fi), (f, fi), and (f, kl), accounting almost 99% of tasks. We particularly focus on failed tasks of failed jobs, corresponding to the following three classes: (f, ev), (f, kl), and (f, fl), which show very uneven distribution and increase the difficulty of an accurate prediction. In the following subsection, we propose a methodology to capture those tasks, as well as significantly reduce their resource consumption.

III. Proposed Methodology

In this section, we describe our methodology to predict job and task classes. We propose an on-line prediction model that can classify jobs in 2 classes, i.e., success (s) and failure (f), and tasks into eight classes, defined by the combination of job classes and four task types, i.e., finish (fi), eviction (ev), fail (fl), and kill (kl). Moreover, we propose a resource conservation policy that can proactively terminate tasks based on the results of the prediction. The goals of our methodology are to 1) accurately predict job and task classes in extremely dynamic and heterogeneous systems, 2) minimize incorrect classifications of successful tasks, and 3) reducing the resource consumption caused by failed tasks of failed jobs.

To such an end, we propose a 2-level NN model that can accurately predict dependent failure patterns of jobs and

tasks. First, job classes are predicted at the first level of the model. Secondly, our model considers only tasks of predicted-to-fail jobs, and predicts their classes at the second level of the prediction model. Based on results of the second level of the prediction, our proposed resource conservation policy proactively terminates those tasks that are predicted-to-fail after a *grace period*, obtained by a combination of *prediction confidences* and task execution times. In this way, the policy is able to decrease the resource consumption, meanwhile minimizing the incorrect terminations of successful tasks. In the rest of section, we first define the quantitative metrics of interests and the used feature sets; then, we describe our proposed 2-level NN model and our proactive resource conservation policy.

1) Metrics of Interests: We are particularly interested in two different metrics: the misclassification rate and the false negative rate. The *task misclassification rate* is defined as the number of misclassified tasks divided by the total number of tasks considered. We consider as *false negative tasks* those finish tasks incorrectly classified by our prediction model, i.e., classified as eviction, fail, or kill, and define the *task false negative rate* as the number of false negative tasks divided by the total number of tasks considered. We want to pinpoint that a high false negative rate has a strong negative impact on Quality of Service (QoS), since terminating successful tasks can greatly impair the user satisfaction. The two metrics can be defined similarly for jobs.

2) Feature Sets: We consider two types of features: *static* features, i.e., related to static information about tasks and jobs collected at arrival time, and *system* features, i.e., the rates of task arrival and throughput of the system. We collect static features at both job and task levels. We use priority and requested resources as static features for tasks. To obtain static features for a job, we consider the average and the standard deviation of all static features among its tasks, in addition to the number of tasks that compose the job. As for system features, we consider task arrival, throughput, and the number of tasks in the system, indicating increment, decrement, and instantaneous state of the system load, respectively. To collect these values, we employ a minimum sampling window, i.e., 5 minutes. We compute the rate of task arrival and throughput for each window and monitor the number of tasks at the beginning of each window. When jobs or tasks arrive at the system, we check system features in the most recent window. Moreover, for each task, we compute the aforementioned load indicators in three variants, including only tasks of (i) the same priority, (ii) lower priority, and (iii) higher priority. We also 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 two recent windows. Finally, we consider both the average and the standard deviation of each aforementioned system feature. In summary, we use a total of 10 job static features, 5 task static features, and 36 system features.

3) On-line Prediction Model: Our proposed on-line NN model uses job, task, and system features collected at the midnight of each day. These features are used to build a new model that can predict job and task classes for the following 24 hours. Each model is trained using data related to the past D days of the workload, where D is the size of the *learning window*. We determine the optimal size D of the learning window during the evaluation process. In the following text, we first provide a background description on NN, then we describe the proposed model.

a) Background on Neural Networks: A *Multilayer Neural Network (MNN)* is an interconnected collection of nodes called *non-linear perceptrons*. A non-linear perceptron is an atomic unit that takes a vector $\mathbf{x}, x_i \in \mathbf{R}, i = 0..N, x_0 = 1$ as input. The unit first computes a linear combination $net = \sum_{i=0}^{N} w_i x_i$ of its input $\mathbf{x}$, where $\mathbf{w}$ is a vector of *weights*, then it applies a *squashing function* $\sigma(net)$ and returns the result as output $o = \sigma(net)$. In a MNN, non-linear perceptrons are interconnected to form a structure composed of three layers, i.e., the input, hidden, and output layers. The *input layer* is formed by the vector $\mathbf{x}$ of real numbers, whereas the *hidden layer* and the *output layer* are formed by H and O non-linear perceptrons, respectively. Each x_i is passed as input to each unit that composes the hidden layer, while the output of the units in the hidden layer are passed as input to each unit in the output layer. In our model, we set $H = 20$, while O is equal to the number of possible prediction classes, i.e., 2 for jobs and 4 for tasks. Since there is exactly one output unit for each class, in the following text we use the term $o_i, i \in \{s, f\}$ to refer to the output value of the success and failure unit for jobs, and $o_i, i \in \{fi, ev, fl, kl\}$ for the output value of the finish, eviction, fail, and kill unit for tasks, respectively. Moreover, we use the *tan-sigmoid* function $\sigma(net) = \frac{2}{(1+e^{-2net})} - 1$ as squashing function for all units in the hidden layer, while we apply the *softmax* function $\sigma_i(net_i) = \frac{e^{net_i}}{\sum_{j=1}^{O} e^{net_j}}$ for the i^{th} unit of the output layer, so that $\sum_i o_i = 1$.

MNNs use a *learning algorithm* to learn the optimal combination of weights for each unit in the hidden and output layers, i.e., the one that minimizes the *error function* of the network. Our model employs the well-know *resilient backpropagation* algorithm as learning algorithm, and the *mean squared error* $E(\mathbf{w}) = \frac{1}{2} \sum_i (t_i - o_i)^2$ as error function, where $i \in \{s, f\}$ for jobs, $i \in \{fi, ev, fl, kl\}$ for tasks, and t_i is the target value of the i^{th} output unit (for a job or task whose real class is j, then $t_{i=j} = 1$ and $t_{i \neq j} = 0$). Essentially, the learning algorithm refines the accuracy of the prediction iteratively, aiming to minimize the mean squared prediction error among all jobs or tasks at each step. Weights are first initialized as random numbers, then features of jobs or tasks in the learning window are passed as input to the network iteratively. In order to cross-validate the accuracy of the prediction, and avoid overfitting biases, we remove randomly 15% of jobs or tasks from the learning window and use them to form a *validation set*. The algorithm terminates when one of the following criteria is met: 1) the misclassification rate evaluated on the validation set increases, 2) the error function fails to decrease for 200 consecutive training steps, or 3) the training step number 500 is reached. We refer the reader to [3] for a more detailed discussion on neural networks.

b) Model Overview: Our model consists of 2 independent MNNs, i.e., the *job MNN*, and the *task MNN*, as shown in Figure 3. As a first step, all jobs are evaluated by the job MNN which constitutes the first level of our model. At this level, the job MNN predicts job classes based on system and static features measured at the job level, and returns the

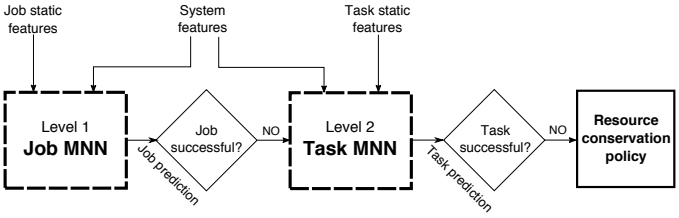


Fig. 3. Flowchart of the proposed 2-level prediction model and resource conservation policy.

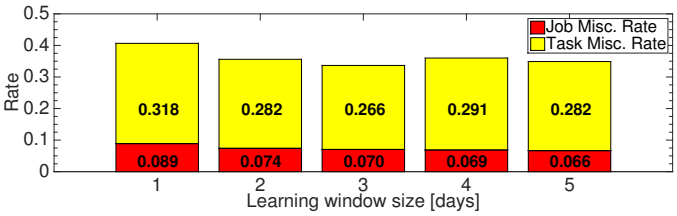


Fig. 4. Misclassification rates of jobs and tasks using different learning window sizes in the training phase.

predicted class of each job as output. The model takes no further action on predicted-as-successful jobs. On the contrary, tasks of predicted-to-fail jobs advance to the task MNN which represents the second level of our model. Here, the model predicts their classes based on system and static features measured at the task level. Similarly to the job MNN, the output of the task MNN is the predicted class of all evaluated tasks. As a final step of our proposed methodology, all tasks classified as fail, evicted, or killed are proactively terminated by our resource conservation policy.

c) Classification and Prediction Confidences: Our prediction model classifies jobs and tasks into different classes based on a prediction confidence. We consider as *prediction confidences* the numerical values obtained as output from the MNNs. In particular, the output $o_i \in [0,1]$ of the i^{th} output unit of a MNN represents the *confidence* of predicting a job or task in the i^{th} class. The more the confidence of one class is close to 1, the more likely the job or task has been correctly classified into that class by the prediction model. A job or task is classified into the class c with the highest confidence, i.e., $c = argmax_i\{o_i\}$, where $i \in \{f, s\}$ for jobs and $i \in \{fi, ev, fl, kl\}$ for tasks. Moreover, our resource mitigation policy leverages prediction confidences to determine the optimal grace period for predicted-to-fail tasks.

4) Resource Conservation Policy: Leveraging the prediction results of the 2-level NN model, we design a delay-based resource conservation policy that terminates predicted-to-fail tasks after a given time interval called *grace period*. The rationale behind our policy is to allow the execution of predicted-to-fail tasks for some time. In this way, false negative tasks still have the chance to execute successfully. On the one hand, setting a long grace period would minimize the number of false negative tasks, decreasing the corresponding resource conservation. On the other hand, a short grace period would lead to high resource savings, at the price of a high false negative rate. By setting the grace period accurately, we can significantly reduce the number of false negative tasks, while still conserving a good amount of resources. As tasks of different classes have very different characteristics and execution time distributions, the policy applies different grace periods to tasks depending on their predicted classes and confidences. For example, tasks classified as eviction usually execute longer than fail and kill tasks before ending their execution; thus, our proposed policy applies a longer grace period for eviction tasks than for fail or kill ones.

The key decision variable in the conservation policy is the length of the grace period. We propose deriving the grace period GP_t for a task t as the weighted sum of the *class grace periods*, denoted by CG_i: $GP_t = \sum_{i \in \{ev,fl,kl\}} \alpha_{i,t} CG_i$, where $\alpha_{i,t}$ is the weight of class i for task t. We obtain $\alpha_{i,t}$ by normalizing the prediction confidences for the task t, i.e., $o_{i,t}$, across the three classes: $\alpha_{i,t} = \frac{o_{i,t}}{\sum_i o_{i,t}}$. Essentially, the grace period of each task is individually decided by its prediction confidences for all three failure classes and the corresponding class grace periods. We determine the optimal class grace period CG_i for each failure class in the evaluation process.

IV. Evaluation

In this section, we evaluate the accuracy of the proposed 2-level NN model on classifying jobs and tasks, as well as the efficiency of the conservation policy in reducing the resource consumption. The evaluation is structured in two different phases. In the *training phase*, we consider only jobs and tasks terminating in the first 14 days of trace and we determine the optimal parametrization of the on-line prediction model and resource conservation policy, i.e., the size of the on-line learning window and the class grace periods. We derive the optimal configuration by extensively experimenting with several different values of these parameters. In the *testing phase*, we apply the optimal model and policy on jobs and tasks terminating in the last 14 days, and evaluate the resulting false negative rate and reduction of resource consumption. In the following, we first present how to parameterize the on-line prediction model and resource conservation policy in the training phase, then we show the results on the testing phase.

1) On-line Prediction Model: Here, we aim to find an optimal size of the learning window D for the proposed 2-level NN model by extensively evaluating several values in the training set. The selection criterion for different values of D is the summation of the job and task misclassification rates. To determine the optimal size of the learning window, we consider five different sizes, i.e., $D = \{1, 2, 3, 4, 5\}$ days, which determine the amount of historical data used to develop the 2-level NN model, and select the one that minimizes the summation of rates. We summarize the job and task misclassification rates with respect to different learning window sizes in Figure 4. One can see that the job misclassification rate shows a decreasing trend with an increasing size of the learning window, starting from an initial value of 0.089 at $D = 1$ and decreasing to a final value of 0.066 at $D = 5$. As for tasks, the misclassification rate has a fluctuating trend across different learning window sizes, with a minimum value of 0.266 at $D = 3$ and a maximum value of 0.318 at $D = 1$. Clearly, the window size that minimizes jobs misclassifications, i.e., 5 days, does not minimize the task misclassification rate. Another observation worth noting is that job misclassification rates are

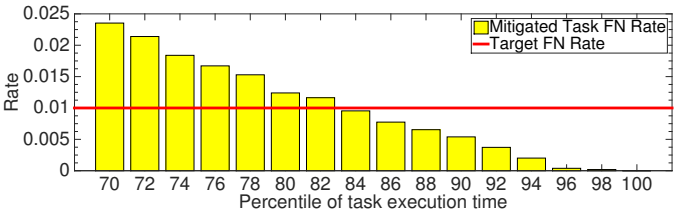


Fig. 5. Mitigated task False Negative (FN) rate using different class grace periods measured in terms of percentiles of task execution times.

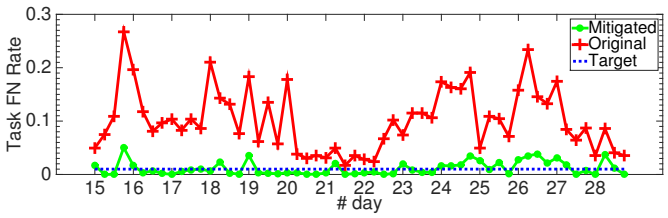


Fig. 6. Testing phase: task False Negative (FN) rate before and after applying the resource conservation policy.

far lower than task misclassification rates in all window sizes. To minimize the summation of job and task misclassification rates, we select $D = 3$ days as the optimal size of the learning window for the testing phase.

2) Resource Conservation Policy: Our resource conservation policy aims to reduce the resource consumption of three classes of failed tasks of failed jobs, minimizing at the same time the risk of terminating misclassified finish tasks. To this end, we apply different grace periods CG_i for the three classes $i \in \{ev, fl, kl\}$ before early termination. Grace periods purposely mitigate the false negative rate by allowing the executions of predicted-to-fail tasks for a given time interval. We define as *mitigated false negative rate* the number of false negative tasks that are terminated by the resource conservation policy divided by the total number of tasks considered. To decide the optimal length of grace periods, we use the mitigated false negative rate as selection criterion as follows. We consider a range of grace periods decided by discrete percentiles of task execution times, and set a *target false negative rate* to 0.01. We summarize the mitigated false negative rates for tasks in Figure 5, with respect to different choices of grace periods. We note that the initial false negative rate obtained from the 2-level NN model is 0.0941 prior to applying the conservation policy. From the figure, one can see that the false negative rate is effectively mitigated by the conservation policy, especially with longer grace periods (higher percentiles of task execution times). Particularly, among all percentiles considered, the 84^{th} is the lowest one that is able to achieve a mitigated false negative rate lower than the target one. As such, we set the three class grace periods to the 84^{th} percentile of the execution time distribution for eviction, fail, and kill tasks, respectively. The exact numerical values are $CG_{ev} = 324$min, $CG_{fl} = 56$min, and $CG_{kl} = 83$min.

In summary, we largely evaluate several values of learning window sizes and grace periods in the training phase. Our results show that the optimal prediction model uses a learning window of 3 days. Moreover, in order to achieve only 1% of mitigated false negatives, the grace periods for the eviction, fail, and killed class are set to 324, 56, and 83 minutes, respectively.

3) Testing Phase: Here, we present the results of applying our 2-level NN model and resource conservation policy on the testing set, using the optimal parameters for the model and policy derived from the training set during the training phase. As a new prediction model is learned and updated every day, we focus on showing the time-varying fluctuations of mitigated false negative rate for tasks and the reduction in the consumption of resources in Figure 6 and Figure 7, respectively.

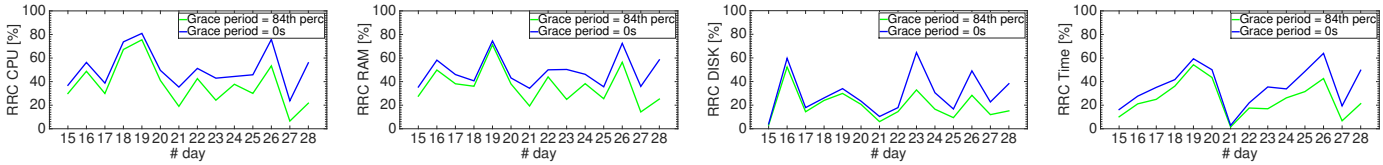
Figure 6 shows the false negative rate for tasks, before and after applying the resource conservation policy. Clearly, the false negative rate is drastically reduced to an average value of 0.012, which is very close to the target value (0.01) set in the training phase. Also, the trend of mitigated false negatives follows closely the original one. Our observations here confirm again that adopting grace periods can effectively minimize the number of false negatives that can seriously undermine the user satisfaction.

In Figure 7, we present the amount of resources conserved by our policy, in terms of CPU, RAM, DISK, and computational time. Prior to explaining the results, we define the metric of *Reduction of Resource Consumption (RRC)*, $RRC = \frac{R_o - R_p}{R_o}$, in order to quantify the resources conserved, where R_o and R_p are the total amount of resources consumed by failed tasks of failed jobs without and with our conservation policy, respectively. We define the resource consumption of each task by multiplying its requested resources and its execution time. We define a RRC for each resource. To highlight the optimality of the grace period chosen in the training phase, we present RRC under two different grace periods, i.e., the optimal one, and no grace period. When no grace period is applied, our conservation policy immediately terminates tasks as soon as they are predicted as failures of failures. For all resources, one can immediately see that our policy can achieve high amounts of resource reduction, with values as high as 80%. We note that terminating failed tasks of failed jobs with no grace period indeed can save slightly higher amounts of resources, but at a price of high false negative rates, as shown by the original trend in Figure 6. In contrast, the conservation policy, using the optimal grace periods, can strike a better tradeoff between mitigated false negatives and resource reduction.

All in all, using the proposed 2-level NN model and resource conservation policy, we are able to save 49%, 45%, 18%, and 26% of CPU, RAM, DISK, and computational time consumed by failed tasks of failed jobs, with an average mitigated false negative rate of 0.012. We thus conclude that the proposed methodology can effectively capture and mitigate failures of failures in big-data clusters, where system and workload dynamics are highly complex and change very frequently.

V. Related Work

Improving dependability of systems and applications is a key requirement for big-data clusters, whose size and complexity are growing endlessly. Although a large body of related



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

Fig. 7. Daily Reduction of Resource Consumption (RRC): a comparison between using the optimal grace period and no grace period in the resource conservation policy.

work analyzes hardware faults [4], [5] in large datacenters, job and task failures have not received much attention by the current literature. In the following, we summarize the related work in two categories: (1) prior analysis on the same Google cluster, and (2) past works on failure modeling and prediction.

1) Google Trace Analysis: This trace [1] has been analyzed by several studies, including general workload characterization [6], [7] and statistical analysis on failures [8], [9]. One of the first studies on the trace is the one conducted by Reiss et al. [6] that pinpoints the high degree of heterogeneity and dynamicity of big-data workloads. Çavdar et al. [7] highlight the execution profile of different task priorities and analyze the root causes of unsuccessful tasks in the system. A recent work by Chen et al. [8] provides a general summary of job and task failures in the trace. In our prior work [9], [10] we analyze the performance impact, patterns, and root causes of different job and task types, `eviction` particularly. Overall, the current literature on this trace provides neither prediction models for failed tasks nor mitigation policies to conserve computing resources.

2) Failure Modeling and Prediction: Several existing studies aim to model failures in big-data systems, focusing on high-performance supercomputers [11], distributed systems [5], virtual machines [12], and storage components [13]. Prior work by Ford et al. [13] proposes a model for data availability in distributed systems using Markov chains. Schroeder et al. [4] show that Mean Time To Failure (MTTF) alone is not a good indicator for disk reliability, while Guan et al. [14] use principal components to predict several types of failures in big-data clusters. In our prior work [2] we propose a predictive model for 2-class job classification that leverages several statistical learning techniques. Compared to our prior work, the current work develops a complete new methodology that classify both jobs and tasks into several classes using neural networks, untangling the complex job-task dependency. While prior art considers mostly static workloads, our proposed on-line classification model is able to predict job and task classes in presence of highly dynamic workloads and time-varying system properties.

VI. Conclusion

Motivated by the significant resource consumption of failed tasks of failed jobs, in this paper we first develop a on-line 2-level NN model that can predict dependent failure patterns among tasks and jobs of big-data applications. Secondly, we design a resource conservation policy that early terminates three classes of failures of failures. To evaluate the proposed model, we use a Google cluster trace as a case study. Our evaluation results show that the combination of the proposed model and policy is able to achieve a consistent conservation of CPU, RAM, DISK, and computational time – as high as 49%, while terminating incorrectly only 1% of successful tasks. In the future, we plan to expand our proposed methodology in order to improve classification results for tasks.

Acknowledgment

This work has been supported by the Swiss National Science Foundation (project 200021_141002) and EU commission under FP7 GENiC project (608826).

References

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

[2] A. Rosà, L. Y. Chen, and W. Binder, "Predicting and Mitigating Jobs Failures in Big Data Clusters," in *IEEE/ACM CCGrid*, 2015.

[3] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[4] B. Schroeder and G. A. Gibson, "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.

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

[6] 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.

[7] D. Çavdar, A. Rosà, L. Y. Chen, W. Binder, and F. Alagöz, "Quantifying the Brown Side of Priority Schedulers: Lessons from Big Clusters," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 3, pp. 76–81, Dec. 2014.

[8] X. Chen, C.-D. Lu, and K. Pattabiraman, "Failure Analysis of Jobs in Compute Clouds: A Google Cluster Case Study," in *IEEE ISSRE*, 2014, pp. 167–177.

[9] A. Rosà, L. Y. Chen, and W. Binder, "Understanding the Dark Side of Big Data Clusters: An Analysis beyond Failures," in *IEEE/IFIP DSN*, 2015.

[10] A. Rosà, L. Y. Chen, R. Birke, and W. Binder, "Demystifying Casualties of Evictions in Big Data Priority Scheduling," *SIGMETRICS Perform. Eval. Rev.*, Mar. 2015.

[11] C. Di Martino, Z. Kalbarczyk, R. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters," in *IEEE/IFIP DSN*, 2014, pp. 610–621.

[12] R. Birke, I. Giurgiu, L. Y. 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] 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.

[14] Q. Guan and S. Fu, "Adaptive Anomaly Identification by Exploring Metric Subspace in Cloud Computing Infrastructures," in *IEEE SRDS*, 2013, pp. 205–214.