Failure Prediction of Jobs in Compute Clouds: A Google Cluster Case Study

Xin Chen*, Charng-Da Lu† and Karthik Pattabiraman*
*Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada.
Email: {*xinchen, karthikp*}*@ece.ubc.ca*
†New York, USA.
Email: *charngdalu@yahoo.com*

***Abstract*—Most cloud computing clusters are built from unreliable, commercial off-the-shelf components. The high failure rates in their hardware and software components result in frequent node and application failures. Therefore, it is important to predict application failures before they occur to avoid resource wastage. In this paper, we investigate how to identify application failures based on resource usage measurements from the Google cluster traces. We apply recurrent neural networks to the resource usage measures, and generate features to categorize the input resource usage time series into different classes. Our results show that the model is able to predict failures of batch applications, which are the dominant jobs in the Google cluster. Moreover, we explore early classification to identify failures, and find that the prediction algorithm provides the cloud system enough time to take proactive actions much earlier than the termination of applications, with an average 6% to 10% of resource savings.**

Keywords: Cloud reliability, Application failure, Failure prediction

I. INTRODUCTION

Cloud systems experience failures due to their large-scale, heterogeneity and distributed nature. Node failures in the cloud may cause the jobs running on them to abort [1]. Also, applications may experience exceptions such as out-of-memory [2] and software bugs. In the cloud, different applications can share resources, and lack of resources may lead to performance degradation and potential application failures.

In this paper, we develop a failure prediction method for application failures of the Google cluster workload traces [3], which contain the workload measurements of more than 12,000 nodes during a one month period. The jobs in the trace range from single-task jobs to multi-task computations [4]. Our goal is to predict if a certain application will ultimately fail, without identifying the underlying reasons, from the perspective of the underlying cloud infrastructure provider. In particular, we do not distinguish the reasons behind the occurrences of application failures, namely performance reasons (e.g., lack of resources) and reliability related hardware/software/network reasons.

Currently, most cloud failures are detected only when they indeed happen. Some prior studies on large-scale system reliability have focused on finding correlations between resource consumption and failure behaviour of applications. Ganesha [5] assumes that fault-free nodes in MapReduce have similar behaviors, and that a deviation from this behaviour indicates a failure. Williams et al. [6] find that a fault possibly manifests as unstable performance behaviors before a failure occurs, thus enabling failure prediction techniques. Besides these algorithms, Ren et al. [2] find that most of the task failures in clouds result from out of memory exceptions, and that job failures are mainly caused by task failures in a commercial cloud trace. While these techniques are useful, none of them can deal with the scale and heterogeneity of large-scale clouds such as the Google cluster.

In this paper, we propose a prediction technique for cloud systems that makes use of the resource usage measures of workloads, to predict job and task failures. The main challenge of using the resource usage time series is to discover features that are indicative of job or task failures. Unfortunately, it is difficult to extract the features directly from the time series data [7]. Instead, we use Recurrent Neural Networks (RNNs) [8], [9] to learn the temporal characteristics of the resource usage measures such as CPU and memory usage. Then we combine trained RNNs with various job/node/user attributes to predict job failures. *To the best of our knowledge, we are the first to predict application (job) failures on the Google cluster dataset and to perform early predictions.*

We make the following contributions in this paper:

- We present a machine learning approach based on recurrent neural networks for predicting job-level and task-level failures. We find that the historical information of jobs from the same user or users affiliated with the same group is essential to achieving high prediction accuracy.
- Our algorithm accurately predicts failures in selected class of failed and finished jobs. For example, our prediction achieves a true positive rate of about 40%, and a false positive rate of 6% when run in a conservative setting.
- We quantify the resource savings achieved by the algorithm on selected batch jobs longer than 1 hour. Using the prediction results, proactive failure management techniques (e.g., killing jobs) provide 6% to 10% of relative resource savings on average.

The rest of the paper is organized as follows. In § II we give more details on the cluster dataset. In § III, we present the overall design of failure prediction. § IV presents the experimental results and analysis of predictions, followed by limitations of this study. The related work is in §V and the paper concludes in §VI.

II. BACKGROUND

In this section, we first describe the Google data sets, with regard to the measured resource metrics and the attributes considered for the prediction. We then introduce the machine learning algorithms we use for prediction.

A. Google Dataset

The Google cluster workload traces [3] are one of the first and few publicly available traces from large cloud systems. Spanning a total of 29 days, around 670,000 jobs and 26 million tasks running on about 12,500 compute nodes are logged in the trace. Figure 1 shows the infrastructure of the clusters.

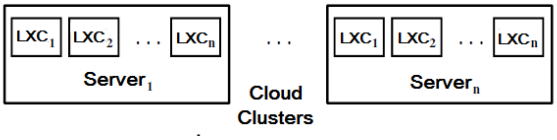


Fig. 1: General Infrastructure

In the trace, a job consists of at least one task, and each task is constrained by scheduling and resource usage limits. The resource isolation and usage measurement are achieved by setting up separate Linux containers (LXC) for different tasks.

A job or task has several possible termination statuses (called "event types" in the trace.) These are: (1) evicted, (2) killed, (3) failed (due to an exception or abnormal condition), and (4) finished (successfully terminated). A job being evicted is a very rare event. Also, no information is provided on whether jobs are killed due to reliability problems. Therefore, we focus mainly on finished and failed jobs in our work. Specifically, we consider two kinds of failures, as shown in Table I.

Failures	Trace Event	Description
Job failure	Job fail event	A job is descheduled due to task failures.
Task failure	Task fail event	A task is descheduled due to a task failure (e.g. exceptions or software bugs).

TABLE I: Definitions of job, task and node failures.

The Google dataset contains periodically profiled resource usage metrics as a time series. Such resource measures consist of CPU usage (average and peak), memory usage (mean, assigned, and peak), page cache (unmapped and total), disk I/O time (mean and peak), disk usage, cycles per instruction, and memory accesses per instruction. All these measurements have been normalized by the respective maximum values measured.

In our earlier work [10], we found that the following attributes were correlated with job failures. Our prediction algorithm will leverage these attributes.

1) **Task Priority** Task priority is one of the scheduling constraints, and determines whether a task is scheduled on a node. General categories are production, batch, and free (low priority batch). The production and free jobs experience much less ratios of failures than the batch jobs.
2) **Task Resubmission** During the life cycle of a job, its tasks can be resubmitted and rescheduled multiple times after abnormal terminations, i.e. failures, evictions or being killed. The ratios of jobs with tasks that execute more than once for failed and finished jobs are 35.8% and 0.9% respectively. In terms of the maximum of task resubmissions, failed and finished jobs have around 400 and 150 resubmissions, respectively.
3) **Resource Usage**: We also found that there are differences in resource consumption between failed and finished task submissions within the same job. At least 34.8% of them show statistically significantly different resource consumptions (mean CPU and memory usage) between failed and finished submissions.
4) **User Profile** The centroids of user profiles groups are correlated with the ratios of job failures to all job termination statuses in the clustering results.

B. Recurrent Neural Networks (RNNs) and Ensemble Methods

Traditional representative techniques, such as the Hidden Markov Models (HMM) and distribution-based methods, have been applied to the time series data in other failure prediction techniques [11]. Different from those data, the Google cluster has a large amount of high dimensional and noisy data that can have dependencies on prior data segments. These properties make the above techniques a poor fit for the Google cluster data. For example, HMMs assume no dependencies exist in the time domain. For distribution-based methods, the heterogeneity or changing mean/variance characteristics make the methods difficult to mimic the data. In comparison, recurrent neural networks (RNNs) [8] can capture the temporal relations in the trace. Further, because RNNs are based on feedforward networks with connections between inputs and outputs, they can handle varying lengths in the time domain. Therefore, we use RNNs in our prediction algorithm.

The Google trace is also extremely diverse in terms of the attributes of the programs, machines and users. Ensemble methods built on single estimators can capture such diversity with robustness [12]. A common selection for the construction is to use the tree-based model as a single estimator with a vector of features. Each estimator can be trained with a random subset of the entire training data. Empirically, ensemble methods tend to generate better results when the data has a significant diversity. Therefore, we use ensemble methods for prediction.

III. Methodology

In this section, we introduce the general framework for the problem, shown in Figure 2.

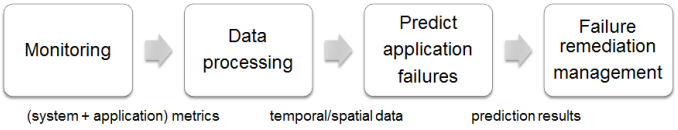


Fig. 2: General Framework

The framework consists of four stages as follows: (1) monitoring and storing the system and application metrics, (2) processing the data to structured formats containing their spatial and temporal information, (3) predicting the failures using machine learning techniques, and (4) failure remediation management based on prediction results. In the Google cluster trace, the system and application metrics are already provided to us. Therefore, we focus on the data processing (2nd) and prediction (3rd) stages. We defer failure remediation based on prediction results to future work.

Data Processing The goal of this stage is to formulate the collected performance data into layered application-centric

structures, which are required by machine learning models. The original tables of system and application metrics cover task resource usage measures and various attributes of the jobs, tasks, nodes and users in separate files. To integrate the data, we join the table files of system and application metrics. Each job is associated with performance data in its all tasks, the job/task/node/user attributes, and the failure data (or termination status mentioned in §II). The outcomes of spatial/temporal data have a two-level format: (1) job-level structured data with the job termination status as the classification target, and (2) task-level structured data with the task termination status as the classification target. At the task level, the resource usage data are organized in the chronological order.

Failure Prediction This stage predicts the termination statuses of tasks and jobs taking the two-level temporal/spatial data as inputs. Figure 3 describes the modules in this stage. The job modelling module trains the predictor, which is composed of RNN based estimator extracting temporal features at the bottom and ensemble methods combining different single estimators at the top. In the test phase, the predictor can be trained from jobs from either all users or one user. Then in the job-level prediction module, the termination statuses of a job and its tasks are predicted. After a certain period (e.g., 1 day), all recent data are retrained in the parameter update module.

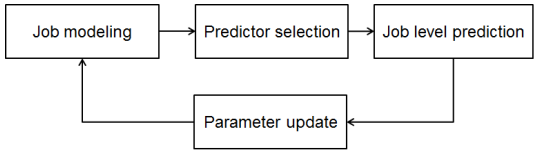


Fig. 3: prediction modules

Recurrent Neural Networks (RNNs)

Given a input sequence of resource usage $x = (x_1, x_2, ..., x_T)$, the standard RNN calculates sequences of states in the hidden layer $h = (h_1, h_2, ..., h_T)$, and sequences of outputs $y = (y_1, y_2, ..., y_T)$. The problem is considered as an instance of the general classification problem. Then the computation has the following iteration equations [8].

$$h_t = H(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \quad (1)$$

$$y_t = softmax(W_{hy}h_t + b_y) \quad (2)$$

where W_{xh}, W_{hh}, W_{hy} are weight matrices, b_h, b_y are biases matrices, H is the hidden layer function (e.g. $tanh$), and $softmax$ is a logistic function (e.g. $tanh$).

The objective function in the RNN problem for a single pair (x, y) is $f = L(\hat{y}, y)$, where L is a distance measurement function between the prediction $\hat{y}$ and the target y. Examples of L include the squared error and edit distance. The overall objective function is the average normalized individual objective function of all data points in the entire set, or practically in the same user.

$$E = \frac{1}{N}\sum_{n=1}^{N} L(\hat{y}_n, y_n) \quad (3)$$

where N denotes the number of sequences, and $\hat{y}_n$ and y_n are the prediction sequence and the corresponding target termination statuses.

The traditional RNN has a serious drawback for data with long-term dependencies: The error-signals could exhibit exponential decay as they are back-propagated through time, which leads to long-term signals being effectively lost as they are overwhelmed by un-decayed short-term signals. To overcome this issue, we need to capture long-term dependencies. We use the Hessian-free optimization [13] to model the temporal connections between hidden states. In this way, we further model the long-term dependencies of resource measurements on prior measurements, and better capture the temporal characteristics of resource usage within an application, particularly those long-running ones.

Prediction of Jobs Prediction is conducted for each job, and the goal is to identify the termination status. Algorithm 1 describes the prediction algorithm.

Input: Two-level data of jobs
Output: Termination statuses of the jobs/tasks
select the ensemble predictor;
foreach *job* **do**
  select the predictors;
  foreach *task in job* **do**
    extract task features/usage time series;
    predict the task termination status;
  end
  generate job feature vector;
  predict the job termination status;
end

Algorithm 1: Prediction Framework

IV. EVALUATION

A. Experimental Setup

The traces are originally stored in comma separated value files of approximate 200GB, and the data attributes are represented by key-value pairs. We read in these traces into a MySQL database for ease of data processing, and store the transformed two-level data of job traces. Then we leverage machine learning packages in Python [14], [15] to implement the prediction modules.

Finer-Grained Selection of Data One of the challenges in the prediction is the heterogeneity of workloads. To reduce the variance between jobs in a category, we divide the entire trace into multiple categories based on the following criteria:
priority: batch, free (i.e. separated from batch for reliability reasons) and production.
job length: short (shorter than 10 minutes), medium (10 minutes 1 hour), and long (longer than 1 hour).
task number: single-task jobs, and multi-task jobs.

In terms of priority, the number of production jobs are much less than that of batch (priority free included) jobs. In addition, some production jobs that consume a lot of resources do not terminate in the monitoring period, we only consider the batch jobs for the prediction. In terms of job length, the number of long/medium jobs is one quarter of the number of short jobs, but long/medium jobs consume much more resources. In terms of task number, the number of single-task jobs is 3 times more than the number of the corresponding multi-task jobs, while the multi-task jobs consume much more resources on average.

Design of the Predictor The predictors are evaluated in the following aspects:

Prediction coverage: The target jobs include long and part of the medium jobs, especially in the categories of heavy resource consumptions. Predicting failures of long jobs can yield higher benefits.
Prediction times: The prediction should be conducted early so that proactive actions can be taken.
Prediction metrics: We define a good predictor as generating high true positive rate (TPR) and low false positive rate (FPR)

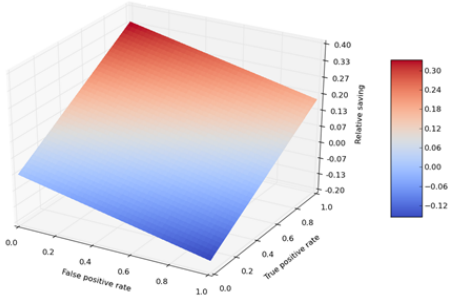
$$\text{TPR} = \frac{\#\text{ successful failure predictions}}{\#\text{ failures}} \tag{4}$$

$$\text{FPR} = \frac{\#\text{ finished predicted as failures}}{\#\text{ finished}} \tag{5}$$

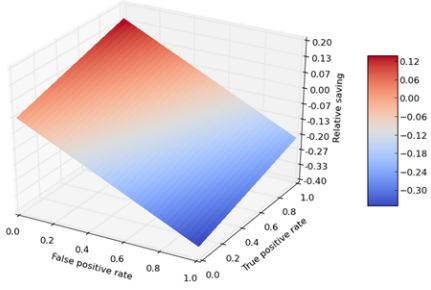
For the classification problems of two classes, TPR and FPR can be denoted by $sensitivity$ and $1 - specificity$.
Resource savings: To estimate the potential resource savings benefited from the prediction, we consider a simple proactive strategy of saving resources, i.e. killing the jobs that are predicted to fail (as permitted by users). Assuming that a job can be killed at most once, we use the following metrics:
R_+: resource saved by stopping failed jobs
R_-: resource wasted by stopping finished jobs
R_{all}: resource consumed by failed and finished jobs
R_{ratio}, which represents the relative resource savings is therefore calculated as:

$$R_{ratio} = \frac{R_+ - R_-}{R_{all}} \tag{6}$$

We select the multi-task/single-task batch long jobs, and predict at the half times of job lengths. Figure 4 plots the potential relative resource savings (i.e. CPU usage) with respect to TPR and FPR.



(a) Multi-task batch long jobs



(b) Single-task batch long jobs

Fig. 4: Approximate relative savings in CPU usage in the predictor designs

Given the same values of TPR and FPR, the relative resource savings can be rather different in the two categories. In the best case, about 32% and 13% of the CPU usage could be saved respectively; and in the worst case, about 15% and 35% of the CPU usage could be wasted. To save more resources, TPR is more important in multi-task long batch jobs, while FPR is more important in single-task long batch jobs. In the predictor design, we can have trade-offs between TPR and FPR by varying classification thresholds, and come up with a conservative predictor (low TPR/FPR) and an aggressive predictor (high TPR/FPR). Separate predictors can be used in different categories to maximize the resource savings.

Neural Network Setup In the training of neural networks, we do not use all the resource usage measures as inputs, but we limit ourselves to 5 popular measures: mean CPU usage, mean memory usage, unmapped page cache, mean disk I/O time, and mean disk usage. Each measure is represented by a class in the input sequences, and thus the inputs have 5 classes of measures at any single time point. The original sampling intervals range from a few seconds to a few minutes. Therefore, we choose time ranges of 15 seconds, 1 minute and 5 minutes, and average the resource usage measurements in these ranges.

For the target sequences, we consider task termination statuses in the failed and finished jobs. To represent the severity of task events, we assign weights of 1, 2, 3, and 4, to the categories finished, evicted, killed, and failed respectively. Note that task failures are labelled with 4, as they have the highest severity.

Experiments on the Workloads Considering the large size of the original data, we conduct the following tests in the first half of the data: (1) We select the failed and finished medium/long jobs, and partition them into training and test sets in the chronological order. At different time slots (quarter, half and the end) within a job, we make the predictions at the task level and job level, and calculate the prediction metrics. (2) We conduct early prediction at the quarter and half times in jobs longer than 1 hour, and then calculate the relative resource savings.

B. Results

Task Level At the task level, we classify the termination statuses of task submissions based on the attributes and performance data. In all the target classes, the status finish is considered as one class, and the other three classes, i.e. evict, kill and fail, are considered as a single class due to the reliability and severity. We evaluate the task level classification in Figure 5.

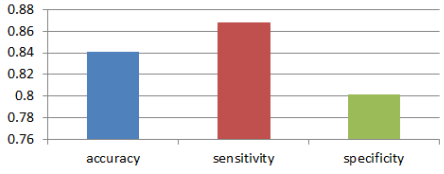
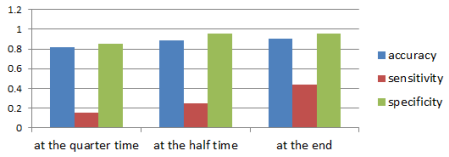


Fig. 5: Task level results of metrics

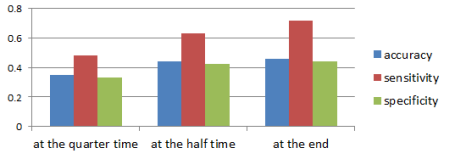
We observe that the classification achieves around 84% of the accuracy, 86% of the sensitivity and 80% of the specificity. With the high true positive rate and low false positive rate, the task level classification serves as the foundation of job level prediction.

Job Level At the job level, we classify the termination statuses of jobs into two classes: fail and finish. Figure 6

shows the prediction results of the conservative and aggressive predictors at different time slots of the jobs.



(a) Conservative predictor



(b) Aggressive predictor

Fig. 6: Job level results of metrics

We observe distinct prediction results from the two predictors at the end of jobs. The conservative predictor has a low FPR of less than 10%, and TPR stays more than 40%. In comparison, the aggressive predictor has around 72% of TPR and 56% of FPR.

The Effects of Selection on Predictors and Predicting Times In Figure 6, the metrics, particularly sensitivity, gradually increase as the prediction time advances, and they do not reveal significant differences across the quarter, half times and the end. It indicates that the jobs have a high possibility of being correctly predicted at the half times if they could be predicted at the end.

We further evaluate the resource savings using the two predictors at the quarter and half times in jobs longer than an hour. Figure 7 shows the relative resource savings of CPU usage, memory usage and task hour in the three most heavy resource consuming job categories: multi-task batch long jobs, multi-task batch medium length jobs, and multi-task free long jobs.

We find that the overall savings in the CPU usage, memory usage and task hours are around 6% to 9% for this predictor at the half times in batch jobs. In comparison, the aggressive predictor either saves or wastes more resources. For example, the aggressive predictor saves about 4.3% and 10% more resources than the conservative predictor at the half times in the multi-task batch long and medium-length jobs. However, it wastes an additional 17% resources in the multi-task batch medium-length jobs.

In all the three job categories, the conservative predictor at the half time is the only predictor that generates positive savings, and can hence be regarded as a stable predictor. Meanwhile, conservative predictors are more friendly to users and job schedulers, as they do not kill jobs unless they are absolutely certain of the job's failure.

User Based Optimization

In this experiment, to reduce the heterogeneity of the training data, we use the previous jobs from the same user to build the model. Only users with more than 1000 jobs are considered for this optimization, while the other users continue to use the model derived from the entire set of users. Figure 8 shows the resource savings of the user-based optimization, compared with the original conservative predictor at the half times.

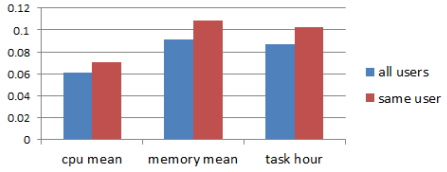


Fig. 8: Resource savings of original predictor and user-based optimization

The overall savings in the user-based optimization, i.e. CPU usage, memory usage and task hour, are around 7% to 10.7% for this predictor at the half times in batch jobs. The extra saved resources are achieved through an additional 11% of increase in the true positive rate at the job level. Since the jobs from the same user may have higher similarity than two random jobs, finer grained categorization of the data may yield better results.

C. Limitations of Our Study

There are two aspects of limitations, one with regard to the trace itself, and the other with regard to our prediction and mitigation strategy.

The Trace

1) The resource consumption is normalized by the maximal values of the resource consumption, and hence some of the original features are lost.
2) Although job failures are identified, the fundamental reasons, i.e. performance reasons or hardware/software related reasons are not distinguished. As a result, we cannot further separate the dataset to provide finer-grained predictions.

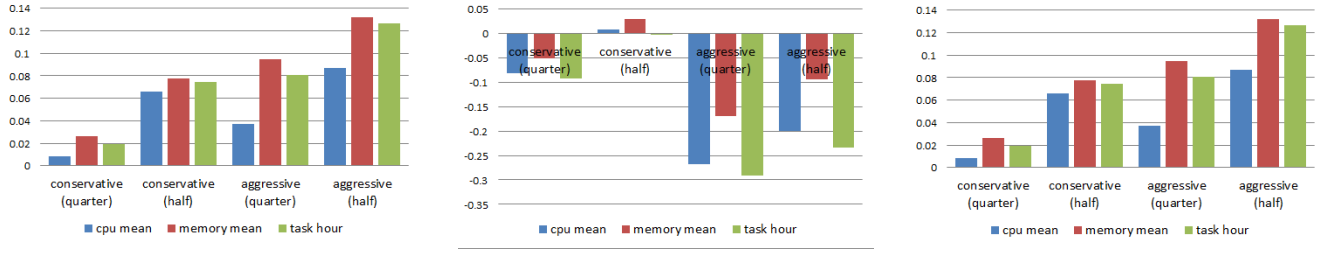
Mitigation Strategy

1) The basic proactive fault management we propose is to simply kill the jobs that are predicted to fail. However, if the prediction is wrong, it wastes resources as the killed jobs would probably be restarted.
2) The failure prediction may not work when the failures happen soon after the faults manifest. It is difficult to predict early enough to avoid the failure in these cases.

V. Related Work

A. Failure Prediction

Online failure prediction based on runtime monitoring is a popular research area. There has been a variety of models and methods that use the current state of a system and, frequently, past experience as well, for example the work by Salfner et al. [11]. Prior failure diagnosis and prediction have been studied in supercomputers and cloud clusters [1], [16], [5], [6]. Liang et al. [1] use tagged logs from the BlueGene machine to discover failures recurrences and correlations between fatal and non-fatal events, and thus predict failures. Using workload traces from *The Grid Workload Archive* project [17], Fadishei [16] et al. find correlations between job failures and attributes including CPU intensity, memory usage, CPU utilization, queue utilization, exit hour and migration of jobs. Pan et al. [5] use the differences in the behavior of faulty and normal nodes in a MapReduce environment to identify failures. However, problems arise when nodes are heterogeneous or few similar nodes can be treated as references. Williams et al. [6] empirically analyze



(a) multi-task batch long (b) multi-task batch medium (c) multi-task free long

Fig. 7: Relative savings of resources (CPU usage, memory usage and task hour) in the groups of high resource consumption

the fault-free and faulty performance data from a replicated middleware-based system, and find that unstable performance is a precursor of failures. They build a black-box method, and predict failure in a window ahead of impending crash failures. In summary, these works predict system failures, or are confined to particular classes of jobs. In contrast, our work is the first to predict application failures in a diverse workload in the cloud.

B. Google Data Set

There have been a number of studies on the Google cluster dataset focussing on the workload characterization and machine utilization. Reiss et al. [4] study the heterogeneity of tasks in the Google dataset. They find that the resources and the tasks executed vary widely. Recently, studies have focused on detecting anomalies in the Google dataset. For example, Guan et al. [18] propose a principal component analysis based algorithm to identify anomalies (failures) by monitoring performance metrics. Their algorithm is essentially built on dimension reduction, which is oriented to their self-collected data with hundreds of dimensions, but show much less accuracy in the Google trace with only 12 dimensions of resource measures. Their goal is equivalent to the task level classification in our algorithm, while we have higher accuracy. More importantly, we predict job failures and propose applying early prediction results to save resources. In recent work, we have characterized the Google data set with regard to its failure behavior and potential for failure prediction [10]. However, we did not consider a specific failure prediction technique in that work, nor do we evaluate the efficacy of different techniques.

VI. Conclusions and Future Work

We present an approach, which builds on the recurrent neural network and the ensemble methods, for predicting failures via various attributes and performance time series data in the Google cluster traces. We successfully predict the termination statuses of tasks and jobs. Experiments show a true positive rate of more than 84% and a false positive rate of 20% at the task level. At the job level, 6% - 10% of resources are saved using early prediction for long batch jobs at the halfway points of their executions.

As future work, we plan to improve the prediction accuracy by fully implementing the parameter update model and adding more features in the learning module. Also, reducing the false positive rate can help the proactive failure management based on prediction results become more effective and save more resources. We would like to extend the prediction framework to general cloud clusters beyond the Google cluster.

Acknowledgements: This work was funded in part by the Natural Science and Engineering Research Council of Canada (NSERC), and the Amazon AWS Education Research Grants.

References

[1] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, "BlueGene/L failure analysis and prediction models," in *International Conference on Dependable Systems and Networks (DSN)*, 2006, pp. 425 – 434.

[2] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou, "Workload characterization on a production hadoop cluster: A case study on taobao," in *Workload Characterization (IISWC), International Symposium on*. IEEE, 2012, pp. 3–13.

[3] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Mountain View, CA, USA, Technical Report, Nov. 2011, revised 2013.05.06.

[4] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 7.

[5] X. Pan, J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Ganesha: Blackbox diagnosis of mapreduce systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 3, pp. 8–13, Jan. 2010.

[6] A. Williams, S. Pertet, and P. Narasimhan, "Tiresias: Black-box failure prediction in distributed systems," in *Parallel and Distributed Processing Symposium. IEEE International*, 2007, pp. 1–8.

[7] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *ACM SIGKDD Explorations Newsletter*, vol. 12, no. 1, pp. 40–48, 2010.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[9] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, pp. 11–24, 2014.

[10] X. Chen, C.-D. Lu, and K. Pattabiraman, "Failure analysis of jobs in compute clouds: A google cluster case study," in *the International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2014.

[11] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, vol. 42, no. 3, pp. 10:1–10:42, Mar. 2010.

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, ser. Springer series in statistics. Springer, 2009.

[13] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 1033–1040.

[14] scikit-learn: Machine learning in python. [Online]. Available: http://scikit-learn.org/stable/

[15] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010.

[16] H. Fadishei, H. Saadatfar, and H. Deldari, "Job failure prediction in grid environment based on workload characteristics," in *Computer Conference, 14th International CSI*, 2009, pp. 329–334.

[17] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema, "The grid workloads archive," 2008.

[18] Q. Guan and S. Fu, "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures," in *Reliable Distributed Systems (SRDS), International Symposium on*. IEEE, 2013, pp. 205–214.