2014 IEEE 25th International Symposium on Software Reliability Engineering

Failure Analysis 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*
†Unaffiliated, New York, USA.
Email: *charngdalu@yahoo.com*

***Abstract*—In this paper, we analyze a workload trace from the Google cloud cluster and characterize the observed failures. The goal of our work is to improve the understanding of failures in compute clouds. We present the statistical properties of job and task failures, and attempt to correlate them with key scheduling constraints, node operations, and attributes of users in the cloud. We also explore the potential for early failure prediction, and anomaly detection for the jobs.**

Based on our results, we speculate that there are many opportunities to enhance the reliability of the applications running in the cloud, such as pro-active maintenance of nodes or limiting job resubmissions. We further find that resource usage patterns of the jobs can be leveraged by failure prediction techniques. Finally, we find that the termination statuses of jobs and tasks can be clustered into six dominant categories based on the user profiles.

***Index Terms*—Job failure, distributions, failure prediction, cloud reliability, anomaly detection**

I. Introduction

Cloud systems experience frequent 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]. One of the main challenges in cloud systems is to ensure the reliability of job execution in the presence of failures [2]. Cloud applications may span thousands of nodes and run for a long time before being aborted, which leads to the wastage of energy and other resources.

In this paper, we conduct a failure analysis 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], [5]. In particular, our goal is to understand the characteristics of job failures in order to improve the dependability of the underlying cloud infrastructure from the perspective of cloud providers. Further, we want to explore the potential for failure prediction and anomaly detection in cloud applications in order to avoid wastage of resources by jobs that ultimately fail. Finally, we would like to understand the effect of job scheduling and node maintenance on the failures.

Many prior studies on large-scale system reliability focus on hardware/software failures and their causes [6], [7], [8]. While these are valuable, they do not provide much insight into failures experienced by end users. Application failures have been analyzed in popular systems such as Hadoop [9], [10] and distributed scientific workflows on Amazon EC2 [11]. However, these studies are limited to MapReduce or scientific computations, and are difficult to extrapolate to generic clouds such as the Google cluster. In comparison, our work focuses on failure characteristics from the jobs' perspective, and covers broader classes of jobs than MapReduce or scientific computations. *To the best of our knowledge, we are the first to perform failure characterization in a large-scale, generic cloud system, from the job and user perspectives.*

The reliability of cloud applications can be affected by the job type, cloud configurations, and the dynamic states of the cloud system. We consider the following four aspects in our analysis of cloud failures: (1) application factors, which are the programs, number of tasks in a job, and the job owners; (2) cloud factors, which are node failures and maintenances; (3) configurations, which include scheduling constraints, and the policy on how many times a failed task can be resubmitted; (4) real-time execution status, which means runtime CPU and memory resource usage. While the Google dataset provides comprehensive logs of each job's resource consumption and failures in the month-long period, it hides specifics about the nature of the job, as well as the physical nodes that the jobs are running on (due to privacy reasons). Therefore, we cannot factor these into our analysis.

We make the following contributions in this paper:

- We analyze the basic patterns and statistical properties of job failures in the Google dataset.
- We correlate the termination of jobs and tasks with job and cloud attributes, such as scheduling constraints, configurations and nodes used. The results show how application failures are affected by job parameters and configurations.
- We identify differences in resource usage between failed and successful jobs with different scheduling constraints and number of tasks. We then investigate how early in the job lifetime such differences manifest, to aid failure prediction techniques.
- We perform clustering on user attributes to understand user-specific job and task failures. We also explore the attributes of users that may be correlated with failures.
- We discuss the implications of our findings on future

1071-9458/14 $31.00 © 2014 IEEE
DOI 10.1109/ISSRE.2014.34

cloud systems that have similar characteristics as the Google cluster.

Our major findings are as follows:

- In the Google cluster workload traces, there is a significant consumption of resources due to failed and killed jobs.
- Job and task failures manifest differently with respect to job and cloud attributes.
  - Task resubmissions in failed jobs are much higher than those in finished jobs on average.
  - Both low and high priority jobs experience on average 3 times as many failures as other priority jobs.
  - Node maintenance and updates are correlated with smaller ratios of task failures on nodes.
- Differences in resource consumption exist between task submissions of failed and finished jobs. For jobs with multiple task submissions, at least 34.8% of the jobs have significant differences between the resource consumptions of failed and finished tasks. In most cases, the differences exist just halfway into a job's execution (for long running jobs).
- User profiles can be clustered into 6 dominant groups, and they are correlated with job failures.

Our study finds that there is significant wastage of resources in the Google cluster due to failed jobs. We also find that there are many opportunities to save resources in the cluster, if that is a desired goal. For example, failure prediction techniques can yield great benefit as they can lead to early termination of jobs (excluding those for debug/test) that are likely to fail ultimately. We find that such prediction schemes are not only feasible, but can yield high accuracy even just halfway into a job's execution (for long running jobs). We further find that user attributes play a major role in the overall reliability of a job, and that users can be clustered into a handful of dominant classes, which can be used for anomaly detection.

The rest of this paper is organized as follows. §II describes the Google cluster dataset. §III characterizes the failures from many perspectives, followed by a detailed discussions of implications and limits of this study in §IV. The related work is in §V and the paper concludes in §VI.

II. Background

A. Google Dataset

The Google cluster workload traces [3] are one of the first and few publicly available traces from large cloud systems (about 12,500 compute nodes over 29 days). The dataset contains the following periodically profiled resource usage metrics: CPU usage (average and peak), memory usage (canonical, assigned, and peak), page cache (unmapped and total), disk I/O time (current and peak), disk usage, cycles per instruction, and memory accesses per instruction. All these measurements are normalized by the respective maximum values measured.

In the trace, every job contains the job name, its resource requirements and the number of tasks in it. A job consists of at least one task, and each task is also constrained by scheduling and resource usage limits. These constraints and limits are present in the trace. The resource isolation and usage measurement are achieved by setting up separate Linux containers (LXC) for different tasks. Around 670,000 jobs and 26 million tasks are logged in the trace.

B. Job/Task Termination Statuses

A job or task has several possible termination statuses (called "event types" in the trace.) These are: (1) evicted, (2) killed, (3) failed, (4) finished, and (5) lost. Evicted means that the system is unable to satisfy the job or task's resource requirements, and hence the job or task is not scheduled. Killed means that the job or task was killed either by the user or by the system administrator, or the job(s) on which it is dependent was terminated abnormally. Failed means that the job or task did not finish execution, and was terminated by an exception or abnormal condition. Finished means the job or task completed execution successfully. Lost means that the record indicating the job termination is missing. Among the above five types, finished jobs are the most frequent (57.6%), followed by killed (40.7%), and failed jobs (1.7%). A job being evicted or lost is a very rare event. Therefore, we focus mainly on finished, killed and failed jobs in this study.

In the paper, we consider three kinds of failures, as shown in Table I. Figure 1 shows the number of job failures over the one month period of the trace. An average of 14.6 jobs fail in a hour, and the minimum and maximum are 0 and 177 jobs, respectively. There is also a rough weekly pattern in the job failures, with failures dipping roughly in weekly intervals, probably due to weekly clean-ups.

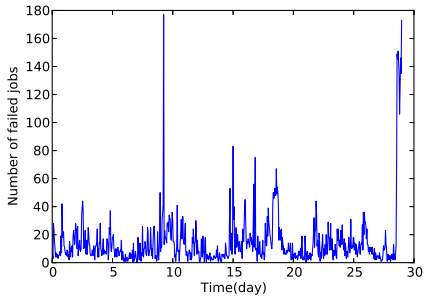


Fig. 1: Failed jobs in the period of one month

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

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).
Node failure	Machine remove event	A node failure leads to the machine being removed from the cluster. Node failures are clubbed together with node maintenance, as they can not be distinguished in the trace.

C. Trace processing

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 analysis. Due to the large scale involved, we deploy databases on Amazon Web Services (AWS) [12] for queries, and leverage machine learning packages in Python [13] for the analysis.

III. Characterization of Failures

In this section we first characterize the durations of failed, finished and killed jobs, followed by the distribution of job resource consumptions. We then study the effects of scheduling constraints on the jobs, and the effect of node maintenance/removal on failures. Further, we investigate correlations between resource usage and the termination status (success/failure) of a job. Finally, we examine similarities among users and user-centric failure characteristics.

A. Basic Failure Distributions

We observe that the job durations of failed, finished and killed jobs follow a heavy-tailed distribution as shown in Figure 2. The majority of jobs terminate within 2000 seconds from start (less than an hour), while the longest failed job lasts for 25 days (almost 29 days including the time to be scheduled.) In addition to the terminated jobs, around 0.5% of the jobs do not terminate in the trace period, and they are not considered in this study.

Among distributions we attempted to fit, we find that the log-normal distribution has the best fit on all the three job termination types, and the parameters are shown in Table II. Our goodness of fit criterion is the Kolmogorov-Smirnov (KS) test. Based on this fitting, we find that finished jobs have shorter lengths than both failed jobs and killed jobs, on average. We speculate three possible reasons for this phenomenon. First, many short jobs are consecutively executed by a few users, and the vast majority of these jobs finish successfully. Second, jobs may hang/freeze and thus get killed after running out of the allocated time or resources. Third, some debug/test jobs can run for a long time before being killed during the development cycle.

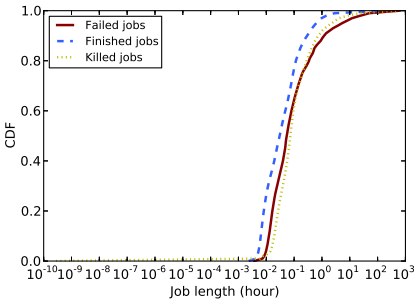


Fig. 2: Distribution of duration of failed, finished and killed jobs

TABLE II: Fitting of log-normal distribution on job duration. μ and σ are the parameters in log-normal distribution, and KS is the maximal distance between distributions in the test.

Type	Mean Duration (Hour)	μ	σ	KS
Failed	2.297	-2.785	1.593	0.06
Finished	0.181	-3.454	1.555	0.11
Killed	1.609	-2.557	1.297	0.06

We also plot the CPU/memory usage of jobs in Figure 3. They also have a heavy-tailed distribution, and follow a log-normal distribution with parameters in Table III. The average CPU and memory consumptions of finished jobs (per second) are around half of those in failed and killed jobs. Overall, the amount of CPU and memory consumed by failed jobs are 2.5 and 6.6 times those consumed by finished jobs. Therefore, we posit that effective failure prediction strategies to prevent resource wastage are needed.

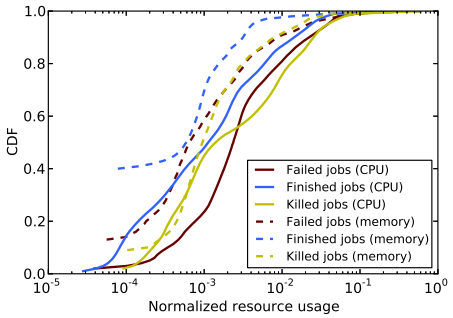


Fig. 3: Distribution of normalized CPU/memory usage of failed, finished and killed jobs. The original units of CPU and memory are core-seconds/second and bytes. Both measurements are normalized by the respective maximal measured value.

TABLE III: Fitting of log-normal distribution on CPU/memory usage.

Type	Mean Resources	μ	σ	KS
Failed (CPU)	0.0080	-6.018	1.426	0.059
Finished (CPU)	0.0047	-6.890	2.088	0.049
Killed (CPU)	0.0089	-6.247	2.149	0.079
Failed (memory)	0.0044	-7.203	1.751	0.118
Finished (memory)	0.0017	-7.702	1.682	0.381*
Killed (memory)	0.0035	-6.878	1.209	0.078

* 0.381 is KS value for the entire distribution, while KS value decreases to 0.082 after removing the biased beginning part.

B. Task Resubmissions

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. A task can also be re-executed if the user so chooses. We examine the effects of task resubmission on the termination statues of tasks and jobs.

In the entire dataset, the ratios of jobs with tasks that execute multiple times for failed, finished and killed jobs are 35.8%, 0.9%, 14.1%, respectively. As expected this ratio is low for finished jobs, which rarely have failing tasks and hence do

not need to reexecute them. Across all categories, we observe that around 76% of the jobs have tasks re-executed at most 4 times. Some systems such as Hadoop MapReduce have limits on the number of task resubmissions, or the user sets a limit on the resubmissions. However, we speculate that in the Google cluster, there is no system-wide limit on resubmission, nor are users mandated to set such limits, and hence it is possible for resubmitted tasks to fail over and over again.

Fig. 4: Numbers of failed, finished and killed jobs with task re-executions. The x-axis is the maximum resubmission time of all tasks in a jobs. A value of 1 means that all tasks in the job are executed once. The y-axis is the cumulative distribution function (CDF).

Figure 4 shows the CDF of task resubmissions on single-task and multi-task jobs for each of the job types. The percentages of failed, finished and killed jobs that consists of multiple tasks (i.e., multi-task jobs) are 17.8%, 4.24% and 53.3%, respectively. In terms of the average number of task resubmissions, the finished jobs have the smallest value, followed by killed jobs, and failed jobs. In each category of jobs, we observe that multi-task jobs have more average resubmissions than their single-task counterparts. Besides, only 0.3% of finished jobs submit tasks more than 10 times, while around 9.5% of failed jobs submit tasks more than 10 times. We also observe that the maximum of task resubmissions in a killed job can be as high as 9062 times (single-task) and 1417 times in (multi-task). In comparison, the maximum of task resubmissions in failed and finished jobs are around 400 and 150 respectively. We speculate that such excessive task re-executions are not useful (except when debugging or testing), and lengthy failed or killed jobs can be preemptively stopped before more resources are wasted.

C. Scheduling Constraints

We also examine jobs by the scheduling criteria and constraints as they may serve different purposes and have divergent behaviours. Jobs and tasks are assigned scheduling classes based on the urgency, the latency sensitivity, and the resource access policies. Production jobs and latency sensitive jobs are likely to be in a higher scheduling class, while non-production or non latency sensitive jobs are likely to be in the lowest class.

Figure 5 shows the distribution of failed, finished and killed jobs for different scheduling classes. The proportion of killed jobs varies across scheduling classes, with the non-latency-sensitive-job class 0 having the highest number of killed jobs. (similar results have also been observed in prior work [14]). However, we find that the ratio of failed jobs to finished jobs is steadily low across most of the scheduling classes, and hence scheduling class does not correlate with job failures. This implies the need to find factors other than scheduling class to characterize failures.

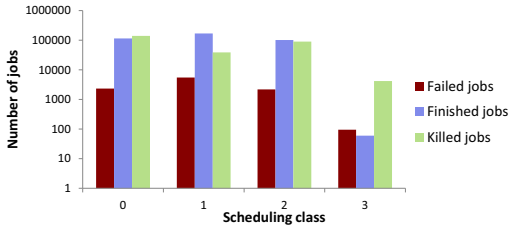


Fig. 5: Failed, finished and killed jobs in different scheduling classes

A more fine-grained categorization of scheduling attributes is the task priority, which determines the nodes assigned to the task, and the turnaround on the task. The priority is associated with only tasks, and not with jobs. The priorities are grouped into five classes [4] ranging from 0 to 11, as described in Table IV.

TABLE IV: Task priorities.

Priority Number	Task Purpose	Level	Note
0-1	Free	Lowest	Resources are rarely charged.
2-8	Batch	Middle	Mainly for batch jobs.
9	Normal production	High	Dominant in production priorities; usually latency-sensitive tasks.
10	Monitoring	High	Monitor the health of other jobs.
11	Infrastructure	Highest	Storage/disk IO services

Normally, all tasks of a job have the same priority. However, 14 out of 925 users have jobs with tasks of two or more priorities. We do not consider such jobs, and group the remaining tasks and their resubmissions by their priorities and termination statuses. The grouping is shown in Figure 6.

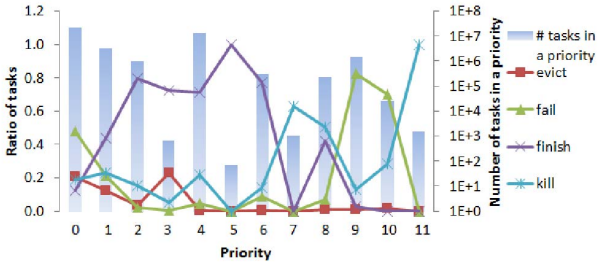


Fig. 6: Task submissions by priority and termination status

As seen in Figure 6, a large number of low-priority tasks are evicted, possibly due to the over commitment of resources. We also see a large number of task failures in the two lowest

priorities. A prominent percentage of tasks of the highest priority are killed, likely because either the requirements of the tasks are not easily fulfilled or because they have hard real-time constraints. On the contrary, most of the middle or batch priorities do not have many tasks that abnormally finished, and they have the highest average ratio of finished jobs in all three priority groups. This is because middle-priority jobs tend to be batch jobs, and we speculate that they often perform routine tasks, and are hence less likely to fail.

The above graph includes task resubmissions, and may hence be biased towards low-priority tasks that have a lot of submissions. To counteract the effect of resubmission, we plot the distribution of tasks after discarding resubmitted tasks in Figure 7. As before, we observe a high number of failed tasks in low- and high-priorities. The ratio of failed tasks in low priorities is low, but the same ratio is three times the average of failure ratios in the middle priorities. This shows that, even ignoring resubmissions, both low- and high-priority tasks are vulnerable to evictions, kills, and failures.

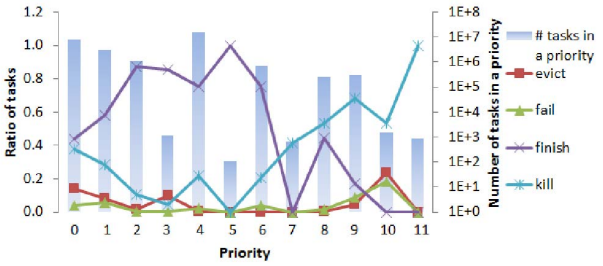


Fig. 7: Task submissions by priority and termination status (excluding resubmissions)

D. Node Failures

Node dependability is an important aspect in understanding the overall dependability of the cloud. However, studying node dependability becomes quite complicated when virtualization allows multiple containers to share a common physical node. The failures may occur due to faults in the hardware, or container software/instances. Unfortunately, the Google trace does not provide design details of the physical machines and containers, and hence it is not possible to track the reliability issues to either physical nodes or containers. However, we can measure: (1) the availability of machines from the perspective of users and (2) the effects of physical node/container reliability on overall task reliability.

In the Google cluster, each machine is identified by a unique ID. Machines may be added and removed from the cluster due to both maintenance or failures. We call the period ranging from an "add machine" event to an "remove machine" event or the end of the trace as a machine cycle. A machine may have multiple cycles if it is repeatedly removed and added.

Figure 8 shows the CDF of the number of machine cycles in the trace. 59.1% of all the machines are never removed from the cluster, and 27% are removed exactly once. More than 99% of machines have less than 6 machine cycles. However, there are some machines that have a high number of cycles. For instance, one machine is removed and added 165 times, and hence has 165 cycles.

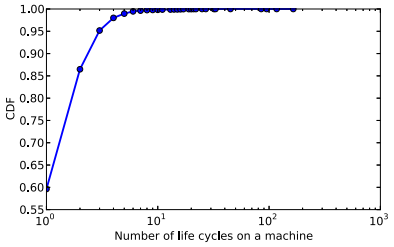


Fig. 8: Distribution of machine cycles.

To calculate the availability of the cluster, the period in a cycle is regarded as the uptime of the system, and the period between two cycles is the downtime. We aggregate the total uptime $E[total\ uptime]$ and total downtime $E[total\ downtime]$ over all machines, and denote the availability by the ratio of total uptime to entire time.

$$Availability = \frac{E[total\ uptime]}{E[total\ uptime]+E[total\ downtime]}$$

We find that the Google cluster has an average availability of 99.82% across all nodes.

To better understand the influence of machine cycles on task failures, we compute the correlation between the average ratio of failed tasks and the number of machine cycles. Only 7 out of the 12,500 machines have more than 30 cycles, and we regard these as outliers. For the remaining machines, we plot the average ratio of failed tasks in each machine cycle in Figure 9. We calculate the Pearson correlation coefficient by comparing the average of failed task ratio with the number of machine cycles. The correlation coefficient is -0.52 (p-value = 0.003), suggesting a medium negative correlation between ratio of failed tasks and the number of machine cycle. We speculate that the machine rejuvenation (removals/additions) may be the cause for the lower ratio of failures.

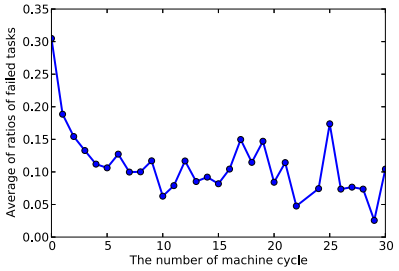


Fig. 9: Average ratio of failed tasks v.s. machine cycles. The x-axis is the machine cycle number. In cycle number k, data points in the kth cycle are chosen from the machines that have no less than k cycles. The y-axis is the ratio of failed tasks.

A machine can also update its available resources and configurations during its life cycle. We plot the average task failure rate in a machine cycle against the number of machine removals in Figure 10, and those machines with frequent updates are plotted in red. As can be seen, high frequency of updates (more than 5 times) occur only on machines with few life cycles (less than 8 times). The task failure rate is at a relatively low level in these nodes despite the fact that removals are not frequent. This is likely because machines that are removed were updated when they were offline. We speculate that updating may be another strategy to enhance the reliability in addition to machine removals.

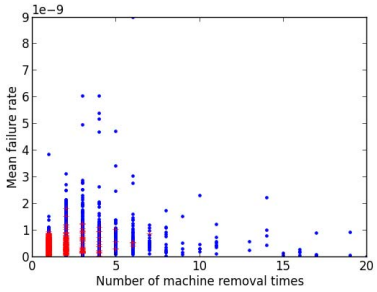
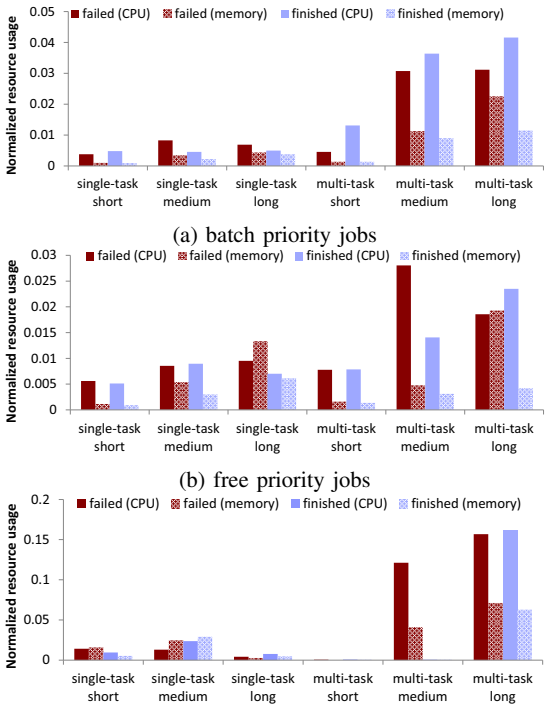


Fig. 10: The failure rate in a machine life cycle. The y-axis is the mean failure rate in a machine life cycle, and the x-axis is the number of machine removals. Every machine life cycle is represented by a blue dot, and the machines having more than 5 updates are plotted in red.



(a) batch priority jobs

(b) free priority jobs

(c) production priority jobs

Fig. 11: CPU and memory usage of jobs under different combinations of job and scheduling parameters.

E. Resource Usage

In this section, we explore the correlations between job resource usage and job failures. We perform the analysis at two levels, i.e., job level and task level.

1) Usage at the job level: We focus on CPU and memory usage in these experiments. Figure 11 shows the CPU and memory usage in failed and finished jobs under a combination of three factors, namely priority class, single/multi-task jobs and job length. The priority classes considered are batch, free and production. The jobs are classified into single-task and multi-task jobs. For the multi-task jobs, the average resource usage is divided by the number of tasks, to ensure that the usage is not skewed by the number of tasks. We classify jobs based on their length as follows: short jobs are shorter than 10 minutes; medium-length jobs are between 10 minutes and 1 hour; long jobs are longer than 1 hour. The combinations of different options add up to a total of eighteen categories.

In all three priority groups, the multi-task jobs generally have more average resource consumption per task, than their single-task counterparts. Because we normalize this on a per task basis, the difference is not because of the higher number of tasks. Further, the difference is most marked between single-task and multi-task jobs that are medium-length or long.

We also observe differences between memory usages of failed versus finished jobs. In general, failed jobs consume slightly more memory than finished jobs for 14 of the 18 categories. Only 4 out of the 18 categories are different: multi-task short batch jobs, single-task medium/long production jobs, and multi-task short production jobs. In contrast to memory usage, CPU usage does not vary as much between failed and finished jobs. In the 18 categories, 12 of them have similar average memory usage, i.e. the ratios of the usage in failed jobs to those in finished jobs range from 0.5 to 2. Seventeen of all the categories have similar average CPU usage. An exception is the short batch group, which accounts for almost one-third of the total jobs, in which failed jobs have lower CPU usage than finished jobs.

To further remove the effects of the heterogeneity among jobs and priority groups, we study the variations among jobs that are resubmitted. Some resubmitted jobs fail, while others finish successfully, and we examine differences in their resource consumptions. The Google manual says that restarting a job will usually generate a new job ID but keep the same job name and user name [3]. So we select all unique jobs with the same job names, and compare the resources consumed by the failed executions and their finished counterparts.

Figure 12 shows the differences between failed and finished jobs that are resubmitted. Failed jobs are found to generally have less CPU and memory usage than the finished counterparts in all three priority groups. Specifically, around 60% to 83% of failed jobs have lower resource usage, i.e., on the left of the red line, in all priorities. For the rest of jobs, finished jobs consume more resources, but the ratios are still close to

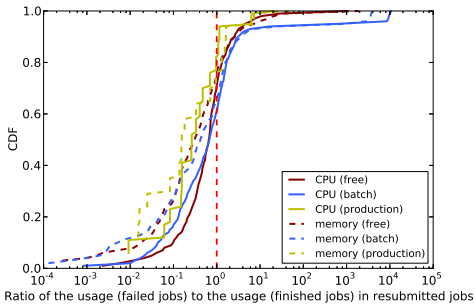


Fig. 12: Ratios of resource (CPU and memory) consumed by failed jobs to that consumed by finished jobs in resubmitted jobs. The resource is calculated by dividing the total job resource by the number of task. The x-axis shows the ratio of average resource in failed jobs to that in finished jobs, and the y-axis represents the cumulative distribution function (CDF). The vertical line in red shows the ratio equals 1.

1. However, a tiny portion of jobs contradict this observation, contributing to the long tails in the distributions. One noticeable example is a series of jobs from one user repeating for more than 27 days, in which the resource consumption of the failed jobs is 1000 times that of the finished jobs.

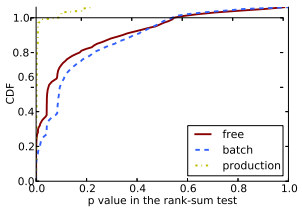
2) Usage at the task level: At the task level, we mainly consider task executions (submissions). We separately gather the resource usage of failed task executions and finished task executions. Different from comparing jobs, all task executions are compared within the same job. The resource usage data are normalized by dividing by the maximum value in a certain resource measurement.

To determine if resource usage samples from two kinds of task executions are significantly different, we use the Mann-Whitney U or the rank-sum test [15], and measure the p-values of the comparison. A p-value less than 0.05 shows that the two samples significantly differ. The rank-sum test does not require samples to be normally distributed, and is hence more widely applicable than other statistical tests.

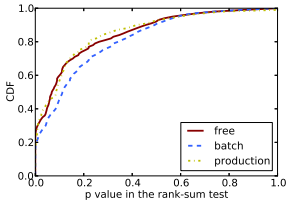
We select the jobs containing failed, finished and killed tasks and study CPU and memory consumption, as inputs to the test. For example, all CPU usage samples in failed and finished executions are the inputs to the rank-sum test to check if distributions of the executions are significantly different in CPU usage consumption. We calculate the p-values of rank-sum tests on normalized resources between each pair of categories. Figure 13 shows the results.

Figure 13a shows the result of running the rank-sum test between failed and finished task executions, for each category. We find that 54.8%, 34.8%, and 93.2% of the p-values are smaller than 0.05 in the free, batch, and production priority classes, respectively. This result implies that most of the production jobs have vastly different resource usage between failed and finished task executions. Batch and free priority classes also have significant differences in CPU/memory usage of tasks in a large portion of jobs.

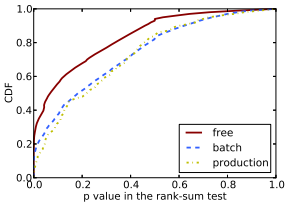
Figure 13b, 13c depict how the killed task executions are



(a) failed and finished executions



(b) failed and killed executions



(c) killed and finished executions

Fig. 13: p-values of normalized resource usage in the rank-sum tests of each pair of categories.

significantly different from the failed and the finished tasks. The average ratios of low p-values in failed/killed test and finished/killed test decrease to around 34% and 28% in all priorities. Production jobs experience the most drops of 43% and 23% respectively.

Early Failure Prediction: The above analysis shows that there are significant differences in resource consumption between failed and finished jobs, suggesting that failure prediction techniques can leverage these differences. We now examine how early in a job's lifetime do these differences manifest, for jobs that exhibit such differences. To explore the possibility of early failure prediction, we examine the differences between failed and finished executions earlier than the termination. Only jobs longer than 10 minutes are selected as we believe that shorter jobs are unlikely to benefit from early prediction. Figure 14 shows the ratios of tests that have p-value of no more than 0.05, and samples are collected from the beginning to 50%, 80% and 90% of the total execution time of the job.

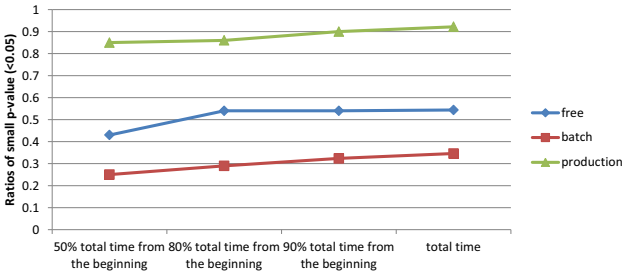


Fig. 14: p-values of normalized resource usage in the rank-sum tests of failed and finished executions. The resources from beginning to 50%, 80% and 90% of the running time in jobs longer than 10 minutes are collected.

We find that the ratio of jobs with small p-values (< 0.05)

for all three priorities remain steady after 50% of the job's execution right until the very end. The decrease in accuracy at half-time compared to the full execution time is negligible. Therefore, for jobs that exhibit differences in their resource consumption, the differences in consumption are significant even halfway into the job.

F. User-centric Analysis

The goal of this analysis is to identify user-specific features that may be correlated with job failures. In the trace, around 700 out of all 933 users have completed jobs and terminated tasks. Further, 334, 397, and 670 users execute failed jobs, finished jobs, and killed jobs respectively.

We would like to have more insights on how user behaviors are correlated with the failures, to understand possible reliability issues for different user classes. However, we do not consider the interactions between jobs from different users or job dependencies, as the Google dataset lacks information about the computation represented by jobs/tasks and the dependencies between jobs. Instead, to get an overall understanding, we cluster the users based on the characteristics and termination status of the jobs submitted by them.

We perform K-means clustering [16] on a user feature vector, consisting of ratios of the failed, finished and killed jobs and tasks. Regarding a user as a data object, the objective is to find the optimal division of data so that a data object is similar to other objects in the same cluster but dissimilar to objects from other clusters. A common measure of dissimilarity s is the Euclidean distance between two data points. The average of s over all points in the dataset, called Silhouette score [17], evaluates how appropriate are the clusters. A Silhouette score close to 1 indicates an excellent clustering. We vary the number of sets from 2 to 10, and find that the best Silhouette score of 0.75 is achieved when 6 sets are selected. The centroids of the 6 clusters and statistics of jobs and resources in each cluster are shown in Table V.

In Table V, the clusters have the following properties.

1) Users in Cluster 1 have more than 87% of jobs being killed, and the ratio of killed tasks is as high as 81%. Further, these users submit the largest average number of jobs in all 6 clusters.
2) Three clusters having many jobs executed are Clusters 2, 3 and 5. The corresponding ratios of killed jobs are greater than 50%, while the ratios of killed tasks is low in these clusters.
3) Users in cluster 3 have the longest median job length of 4448 hours, and 42.5% of these jobs fail at the end. This leads to potentially significant wastage of resources.
4) Cluster 4 has the highest ratio in the finished jobs/tasks among all the clusters, of about 83%. Further, each job has a median number of 2755 tasks in this cluster.
5) Cluster 5 contains the most evicted jobs and tasks among all clusters. The sixth cluster has a balanced ratio between the failed jobs and finished jobs.

Table V also shows the attributes of resource usage for each cluster. The resource usage statistics concern the average CPU and memory usage measurements per unit time. CPU usage is found positively correlated with ratios of finished jobs/tasks, but no obvious pattern applies to memory usage. We also inspect the user behaviour of executing production jobs and batch jobs, and select users who run more than 20% of production jobs in the entire period. The vast majority of these users are grouped into Cluster 1, 3 and 5. Cluster 1 in particular has about 60% of the production users. This implies strong correlations between user behaviours of submitting jobs and the outcomes of application reliability. This similarity can be potentially leveraged by anomaly detection and failure prediction systems.

IV. Discussion

In this section, we discuss the implications of our results, followed by the threats to the validity of the study.

A. Implication of the Results

Much of this section is based on our speculation and anecdotal experience, and is not backed up empirically.

Our analysis results are useful for failure-aware resource provisioning [18], failure prediction, and resource provisioning policies. Such policies have been used in failure-aware scheduling and energy-aware scheduling [19] to mitigate the effects of failed and killed jobs. We find that finished jobs have much shorter running times and consume fewer resources than failed and killed jobs in §III-A. This implies that a lot of resources may be wasted on jobs that do not finish, except those that are for debugging or testing purposes. Nevertheless, this indicates the need for early failure prediction at the infrastructure provider level.

We also found that the termination statuses of jobs are influenced by the job's pre-launch attributes (namely, priority and resubmission rule) in §III-B. For example, failed and killed jobs have high number of resubmissions. To save resources, it may be a good idea to limit the number of job resubmissions (for some classes of jobs) if a job is predicted to fail or terminate unsuccessfully, especially for automated resubmissions.

Another issue is that low-priority jobs contend for resources with high-priority jobs, making it more likely for high-priority jobs to possibly fail and thus waste resources. Further, both low- and high-priority jobs experience high failure ratios (§III-C), and hence there is a need for a scheduler that can adjust job priorities based on their failure histories.

Although Google does not disclose how they maintain and update machines in the cluster, we find that machines and containers that experience removals or updates are less prone to failures (§III-D), suggesting that these operations improve reliability. This is similar to the idea of software rejuvenation [20], but at the container level.

We also observed correlations between the resource consumption of jobs and their propensity for failure in §III-E. While these correlations depend on the job's priority class and whether the job is single- or multi-task, there are significant differences between the resource consumptions of failed and

TABLE V: K-means clustering on users profiles. The features for clustering are the ratios of evict, fail, finish and kill events in both jobs and tasks. The statistics of job attributes and resource usage are in average, and # represents the number of a variable. In the user attributes, a user is called production user if the production jobs account for more than 20% of its all jobs.

Cluster	Centroids of Features								Statistics						
	Ratios of Jobs				Ratios of Tasks				Job Attributes			Resource Usage		User Attributes	
	Evict	Fail	Finish	Kill	Evict	Fail	Finish	Kill	# Job	# Tasks/Job	Length	CPU	Memory	# Production User	# User
C1	0	0.0604	0.0633	0.8763	0.0705	0.0554	0.0644	0.8097	443	536.8	790.98	0.00076	0.00263	56	224
C2	0	0.0498	0.316	0.6341	0.0645	0.0398	0.6816	0.214	238.79	1525.85	1035.57	0.01376	0.0039	2	184
C3	0	0.425	0.0741	0.5009	0.0597	0.7307	0.0499	0.1597	281.18	227.81	4448.71	0.0034	0.00713	16	63
C4	0	0.0444	0.8367	0.1188	0.0439	0.0441	0.8012	0.1108	18.43	2755.82	705.96	0.00545	0.00526	2	84
C5	0.0079	0.1846	0.0664	0.741	0.7	0.0559	0.08	0.164	349.6	69.74	753.32	0.0019	0.00614	13	126
C6	0	0.0395	0.8127	0.1477	0.1221	0.2946	0.1818	0.4015	28.82	952.47	664.8	0.00176	0.00998	5	19

finished tasks, both in CPU and memory consumption, so a good failure prediction could help the resource scheduler allocate the resources differently among predictably faulty and successful jobs. Further, when these correlations manifest, they do so as early as 50% into the job's run time, thereby indicating the potential for early failure prediction for long jobs. We define a threshold of 1 hour, to filter long jobs and apply the failure prediction algorithm. Such jobs can last anywhere from a few hours to a few days, so one can wait till the threshold, and still get significant resource savings.

Finally, we find that job failure behaviour can be clustered into six categories based on the users submitting the jobs, and that each category has distinctive patterns in terms of job attributes and resource consumption in §III-F. This information can be used in anomaly detection, for example, to detect jobs that deviate significantly from the characteristics of their categories and perhaps terminate them early. This would allow more efficient resource utilization in the cluster.

B. Threats to Validity

Our study focuses on the Google cluster, and hence may not be generalizable to other cloud infrastructures. This is an external threat to the validity. One way to mitigate this threat is to study other cloud infrastructure failures. However, there is no publicly available failure data from real-world cloud deployments on the same scale as the Google cluster.

The main internal threat to validity is that the Google dataset is both incomplete and anonymized (out of privacy concerns). In particular, there are four limitations:

1) It is not clear that who the users are, what their workflows are, and why the users were running the jobs. Therefore, it is difficult to say anything about the effect of failures on the overall user experience.
2) A job can fail either because of performance reasons (e.g. lack of resources) or reliability reasons (hardware/-software/network failures) or simply for testing/debugging purposes. The traces do not have enough information to infer the job failure causes.
3) The dataset does not have program or application information, such as whether the programs were MapReduce jobs. It does not have any information about the job schedulers, or other software running on the nodes.
4) The resource consumption is normalized by the corresponding maximum values, and the raw values are not provided. Hence, we cannot understand the reasons behind why certain consumption patterns are correlated with failures.

To mitigate the above threat, we need more information about the traces, but unfortunately, this is not publicly available.

There is a construct threat to validity in that we have assumed that resource conservation is a desired goal for the users of the cluster. However, this need not be the case as the cluster may be used purely for debugging or testing tasks, where job failures are the expected behaviour. As such, this threat can be mitigated if we knew what the cluster is used for, but this is not the case.

Finally, there is a threat regarding the reproducibility of the results and our conclusions. We have made all our data publicly available, and also the scripts used to generate them, to mitigate this threat[1].

V. Related Work

We classify related work into three broad categories as follows.

Failure analysis. Prior studies characterize failures in supercomputers and clouds from the perspective of system failures [6], [7] and application failures [9], [21]. ElSayed et al. [6] perform a comprehensive statistical analysis on supercomputer logs from Los Alamos National Labs. They also explore the impact of environment issues on failures. Vishwanathan et al. [7] explore the hardware reliability of clouds. They find that disks are the main culprit in node failures. Unlike our work, these studies focus on hardware reliability rather than job failures, which can be caused by hardware, software and configuration failures.

Kavulya et al. [9] analyze logs from Hadoop applications and characterize their job patterns and the failure causes. Ren et al. [10] study the logs collected from Hadoop clusters running e-commerce applications. In contrast to these datasets, the Google dataset has a more diverse workload, and hence our findings are applicable to a broader range of cloud applications.

Failure characterization and prediction Prior results on failure analysis or characterization have been applied to failure diagnosis and prediction [22], [23], [24]. Using workload traces from *The Grid Workload Archive* project [25],

[1] http://www.ece.ubc.ca/~xinchen/cluster_trace/data.zip.

Fadishei [22] 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. [23] use the differences in the behavior of faulty and normal nodes in a MapReduce environment to identify failures. Williams et al. [24] empirically analyze the fault-free and faulty performance data from a replicated middleware-based system, and find that unstable performance is a precursor of failures. While these works have all investigated the relationship between resource consumption and job failures, they have been confined to particular classes of jobs. In contrast, our work is the first to explore such correlations in a diverse workload in a production cloud.

Oliner et al. [26] demonstrates that failure-aware scheduling can be effective even with modest prediction accuracy. They show that improved scheduling of parallel jobs has a significant impact on the job response time and overall system utilization. Liu et al. [27] focuses on adjusting the placement of active or running jobs in response to failure prediction, and proposes an application-level job migration and processor swapping approach to diminish the impact of failures. Our work is orthogonal to the above techniques, as it deals with failure characterization, but can facilitate failure aware scheduling and placement.

Google cluster dataset There have been a number of studies on the Google cluster dataset focussing on the workload characterization and machine utilization. Liu et al. [14] perform a statistical analysis of node, job and task level workload with respect to resource utilization. Reiss et al. [5] study the heterogeneity of tasks in the Google dataset. They find that the resources and the tasks executed vary widely. Khan et al. [28] propose an accurate characterization that can faithfully reproduce the performance of historical workload traces in terms of key performance metrics, such as task wait time and machine resource utilization. Zhang et al. [29] propose a model for runtime task resource usage that is able to reproduce aggregate resource usage and scheduling delays. They find that using the mean and coefficient of variation within each task can generate synthetic workload traces, reproducing accurate resource utilizations and task waiting time. Di et al. [30] compare the differences between the Google data center and a Grid system. They find that the Google dataset exhibits finer granularity resource allocation with respect to CPU and memory than the selected Grid system. The main difference between these papers and ours is that none of them study failures or failure-related attributes.

Recently, there have been three studies on understanding failures in the Google dataset. We explain the differences between these papers and ours below.

Di et al. [31] use job-specific information and the termination statuses of tasks, and apply the K-means clustering to characterize the jobs. However, their analysis is based on logical job names, which are not guaranteed to be unique. The application properties may be dominated by a few jobs, and hence bias the results. In contrast we use job IDs which are guaranteed to be unique, and provide a higher coverage on characterizing jobs and tasks. Further, rather than simply clustering task events to get centroids of clusters, we correlate the clusters of failures with user profiles, and we consider job events as well. Finally, job attributes such as priorities, resubmissions and run time are not considered in their paper, while they are considered in ours.

Guan et al. [32] use principal component analysis on the task resource consumption to identify the features most likely to influence failures. They find that the average correlations of the raw resource usage to the failures are around 0.07 in all tasks. In contrast, we perform finer grained analysis on different classes of jobs and resources, and we find much higher correlations and more significant differences between failures and successful terminations. For example, in our analysis, at least 34.8% of the jobs have significant differences between the resource consumptions of failed and finished tasks.

In very recent work, Garraghan et al. [33] study the node and task failures' statistical distributions. However, distributions are not enough to characterize machine and task failures, as the workload is highly diverse. In contrast, we use job and cloud system attributes to understand the correlations between job failures and attributes. They also label the node maintenance as failures. However, the work by Reiss et al. [5] has shown that node maintenance is mostly planned downtime, and hence different from failures. Finally, they do not consider the correlations between resource consumption of the jobs and their failures, while we do.

VI. Conclusion and Future Work

This paper investigates the characteristics of failed and killed jobs in Google's production cloud system. We characterize failures of jobs with respect to their attributes, and study the effects of attributes, such as priority, task submissions, and resource consumptions, on job failures. Failed and finished jobs and tasks have different characteristics of resource usage, and these differences have a high probability of manifesting well before the jobs' end.

Our study points to the importance of failure prediction for resource provisioning and scheduling in compute clouds. In the future, we will leverage the findings of this paper to develop anomaly detection and early failure prediction algorithms for better cloud utilization and reliability. In addition, we plan to extend our study to a wider range of cloud systems.

Acknowledgements

This work was funded in part by the Natural Science and Engineering Research Council of Canada (NSERC) through the DIVA network and the Discovery Grants Program. The experimental environment was supported by Amazon AWS Education Research Grants, and the Canada Foundation for Innovation (CFI). We would like to thank Marcus Carvalho, Lauro Costa and Sathish Gopalakrishnan for their helpful suggestions. We also thank the anonymous reviewers and the shepherd of ISSRE 2014 for their insightful comments.

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] C. Pham, P. Cao, Z. Kalbarczyk, and R. K. Iyer, "Toward a high availability cloud: Techniques and challenges," in *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*. IEEE, 2012, pp. 1–6.

[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. Posted at URL.

[4] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale: Google trace analysis," Intel science and technology center for cloud computing, Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. ISTC–CC–TR–12–101, Apr. 2012.

[5] 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.

[6] N. El-Sayed and B. Schroeder, "Reading between the lines of failure logs: Understanding how hpc systems fail," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, 2013, pp. 1–12.

[7] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 193–204.

[8] F. Dinu and T. Ng, "Understanding the effects and implications of compute node related failures in hadoop," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*. ACM, 2012, pp. 187–198.

[9] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010, pp. 94–103.

[10] 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), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 3–13.

[11] T. Samak, D. Gunter, M. Goode, E. Deelman, G. Juve, F. Silva, and K. Vahi, "Failure analysis of distributed scientific workflows executing in the cloud," in *Proceedings of the 8th International Conference on Network and Service Management*. International Federation for Information Processing, 2012, pp. 46–54.

[12] Amazon web services (aws) - cloud computing services. [Online]. Available: http://aws.amazon.com/

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

[14] Z. Liu and S. Cho, "Characterizing machines and workloads on a google cluster," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, 2012, pp. 397–403.

[15] J. McKean and T. Hettmansperger, *Robust nonparametric statistical methods*. CRC Press, 2011.

[16] 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.

[17] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[18] B. Javadi, J. Abawajy, and R. Buyya, "Failure-aware resource provisioning for hybrid cloud infrastructure," *Journal of parallel and distributed computing*, vol. 72, no. 10, pp. 1318–1331, 2012.

[19] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem, "Energy aware scheduling for distributed real-time systems," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*. IEEE Computer Society, 2003, pp. 21–2.

[20] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa, "Workload-based software rejuvenation in cloud systems," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1072–1085, 2013.

[21] K. Ren, Y. Kwon, M. Balazinska, and B. Howe, "Hadoop's adolescence: An analysis of hadoop usage in scientific workloads," *Proc. VLDB Endow.*, vol. 6, no. 10, pp. 853–864, Aug. 2013.

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

[23] 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.

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

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

[26] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam, "Fault-aware job scheduling for bluegene/l systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 64.

[27] Y. Li, P. Gujrati, Z. Lan, and X.-h. Sun, "Fault-driven re-scheduling for improving system-level fault resilience," in *Parallel Processing, 2007. ICPP 2007. International Conference on*. IEEE, 2007, pp. 39–39.

[28] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, 2012, pp. 1287–1294.

[29] Q. Zhang, J. Hellerstein, and R. Boutaba, "Characterizing task usage shapes in googles compute clusters," *Proceedings of LADIS*, pp. 2–3, 2011.

[30] S. Di, D. Kondo, and W. Cirne, "Characterization and comparison of cloud versus grid workloads," in *Proceedings of the 2012 IEEE International Conference on Cluster Computing*, ser. CLUSTER '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 230–238.

[31] S. Di, D. Kondo, and F. Cappello, "Characterizing cloud applications on a google data center," in *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 2013, pp. 468–473.

[32] Q. Guan and S. Fu, "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures," in *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*. IEEE, 2013, pp. 205–214.

[33] P. Garraghan, P. Townend, and J. Xu, "An empirical failure-analysis of a large-scale cloud computing environment," in *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on*. IEEE, 2014, pp. 113–120.