[Technical Report CSRG-627 (University of Toronto)]

How reliable are large-scale jobs in parallel clusters?

Nosayba El-Sayed Bianca Schroeder
Department of Computer Science, University of Toronto
{nosayba, bianca}@cs.toronto.edu

I. Introduction

As large-scale platforms continue to grow in scale and complexity, the applications running on them are increasingly being prone to errors and failures. A recent study by Snir et al. [9] predicts that large parallel applications running on an exascale machine may abort as frequently as once every 30 minutes. Minimizing the rate at which large-scale jobs fail is especially critical for long-running applications that can take up to weeks or months to finish executing successfully [8]. Emerging challenges, such as the increasing complexity of parallel application workflows [5] or the rising silent data corruption (SDC) rates in large-scale platforms [6], require substantial improvements in both system-level and application-level resilience.

In this technical report we are interested in answering the question of what makes jobs fail in large, parallel clusters. Towards this end, we use workload traces collected at real world systems to study the effect of different factors on *job reliability* and to identify which ones are predictive of job failures in parallel applications. We investigate factors such as job configuration parameters (e.g. the degree of parallelism, priority, etc), job resource requests and constraints, job resource utilization (e.g. memory and CPU usage), and others.

We begin by describing the workload traces included in our study in Section II. Section III focuses on characterizing job failures in these clusters by using the traces to quantify the correlation between a diverse set of factors and a job's exit status. Finally, in Section IV we study how to predict job failures in parallel clusters using different classification techniques and evaluate the quality of our predictors using real world traces.

II. Description of workload traces

We rely on workload traces made available by three different sources: Google, Carnegie Mellon University (CMU), and Los Alamos National Labs (LANL):

(a) Google: For one of Google's large, multi-purpose compute clusters consisting of 12,000 machines, traces of all jobs submitted to the cluster during 29 days in May, 2011 are made publicly available [10]. Each job is comprised of one or more *tasks*, and each task is associated with a set of resource requirements. A task is assigned to run on a single machine, and data on task resource consumption is made available, such as CPU, memory, and disk usage (aggregated over 5-minute intervals). All values of usage data are normalized by the maximum value reported in each category, and fields on users and application names are obfuscated. The traces include data on when each job or task was submitted, scheduled, and finished (or was aborted). The exit status of each job (and task) attempt is included in the traces and can be one of the following: *failed*, *killed*, *evicted*, or *finished*.

Jobs were submitted by hundreds of users, where a user can be a Google engineer or a service [7], and their tasks were assigned *priorities* from 0 (lowest) to 11 (highest). According to a paper that accompanied the release of the trace [7], priorities 0 and 1 are labeled as gratis (free); priorities 2–8 are speculated to be dominated by batch jobs and MapReduce jobs; priority 9 is 'normal production' jobs; priority 10 represents 'monitoring' jobs; and priority 11 is labeled 'infrastructure'. In our study, we focus on two priority groups: production jobs (priority 9) and batch jobs (priorities 2–8).

(b) CMU OpenCloud Traces: OpenCloud [3] is a 64-node research cluster at Carnegie Mellon University used by CMU researchers and managed by the Parallel Data Lab [2]. The cluster runs Apache's Hadoop [1] distributed file system, and had different types of applications running on it during the trace collection period from areas such as computational biology, natural language processing, image processing, and machine learning [3].

The traces were collected over a period of 30 months and include two types of files: job configuration files, and job history logs. The configuration files contain information on the parameters set by the users for each submitted job, such as (but not limited to): the size of virtual memory for *map* and *reduce* tasks; the maximum number of attempts per task before the framework gives up on it; and boolean flags for whether or not to enable speculative execution. The history files contain data on job submission time, launch time, and finish time; the exit status of a job (*succeeded*, *failed*, or got *killed*); the number of map/reduce tasks in the job; the number of tasks in the job that completed successfully; and the number of non-successful task attempts.

(c) LANL Job Traces: LANL made publicly available three years worth of job traces for one of their HPC systems (system with ID 20 on their webpage [4]), which comprises of 256 nodes. The HPC cluster was used by scientists, typically to run CPU-intensive scientific simulations. Around 400 users submitted approximately 300,000 jobs in total to cluster during the trace collection period.

The workload traces contain data on job submission time, launch time, and exit time; the number of processors requested for this job; the IDs of the nodes that the job was assigned to; and the final status of the job. The table below summarizes the possible exit statuses for a job submitted to the LANL cluster:

LANL Job Status	Description
`finished`	All processes are scheduled
`failed`	One or more of the nodes running this job crashed
`aborted`	User aborted job (Ctrl/C)
`killed`	An application process was killed by a signal
`syskill`	Job was killed by an administrative user (either due to maintenance or due to a problem).

TABLE I
DESCRIPTION OF EXIT STATUS MEANING IN THE LANL JOB TRACES.

We now look at the basic statistics and overall success rates for the jobs submitted to the three clusters in our dataset. Table II below provides an overview of the clusters and their job traces, along with the breakdown of the job exit status. (Note that as previously indicated, for the Google traces we will be studying jobs that are categorized as ‘production’ jobs separately from jobs categorized as ‘batch’ ones.)

Data Source	Data Timespan	Number of Nodes	Number of Users	#Scheduled Jobs	%Success Jobs	%Failed Jobs	%Killed or Aborted Jobs	%Other
Google Multipurpose Cluster (Prod Jobs)	29 days	12K	265	43K	83%	1.66%	15.25%	0.01%
Google Multipurpose Cluster (Batch Jobs)	29 days	12K	227	349K	63%	1.27%	35.5%	0.003%
CMU OpenCloud Cluster	883 days	64	157	78K	88.8%	8.11%	3%	0.04%
LANL HPC System#20	1,022 days	256	446	290K	67.5%	0.62%	30.7%	1%

TABLE II
OVERVIEW OF THE WORKLOAD TRACES AND BASIC JOB STATISTICS FOR THE CLUSTERS IN OUR DATASET.

We find that job success rates vary across the different clusters in our data and fall within the 60–90% range. Unsuccessful job submissions were dominated by killed (or aborted) jobs followed by failed jobs, with the exception of the OpenCloud cluster where an opposite trend was observed. We next take a closer look into jobs that terminated unsuccessfully, with the goal of developing a better understanding of why they failed and whether certain factors are more likely to be correlated with their failure.

III. UNSUCCESSFUL JOBS: TRACE-BASED ANALYSIS

In this section we use the traces to study the correlation between several factors and job reliability in large clusters, including: the degree of parallelism, job configuration parameters, fault-tolerance parameters, job durations, and resource utilization and constraints.

A. Degree of Parallelization (DOP)

We begin by studying the relationship between the degree of parallelism of a job and its final status. In Google's cluster, a job consists of one or more tasks that typically execute the same binary with the same resource requirements and scheduling constraints (e.g. priority, scheduling-class, etc). Applications that need to run different types of tasks will usually execute them as separate *jobs* [10]. For example, MapReduce applications would execute masters and workers as separate jobs. Generally, multi-task jobs are meant to have their tasks run simultaneously, where a single task can be running on a single machine at any point in time. A configuration parameter is available where a user can indicate if tasks must execute on *different* physical machines.

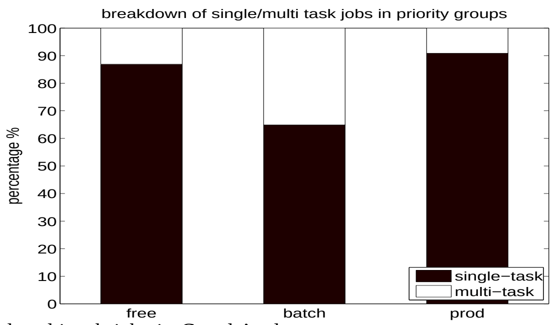


Fig. 1. Breakdown of single-task and multi-task jobs in Google's cluster.

Figure 1-(a) shows the breakdown between single-task and multi-task jobs for each priority group in Google's cluster. We find that production jobs are dominated by single-task jobs (90%), whereas 65% of batch jobs are single-task. Multi-task jobs, however, consume significantly more compute cycles in Google's cluster: 98% and 99% of task-minutes consumed by production jobs and by batch jobs, respectively, belong to multi-task jobs. To study whether the success rate of submitted jobs varied between single-task and multi-task jobs in Google's cluster, Figure 2 plots the breakdown of jobs' exit status (i.e. completed, failed, or got killed) for single-task and multi-task jobs, separately:

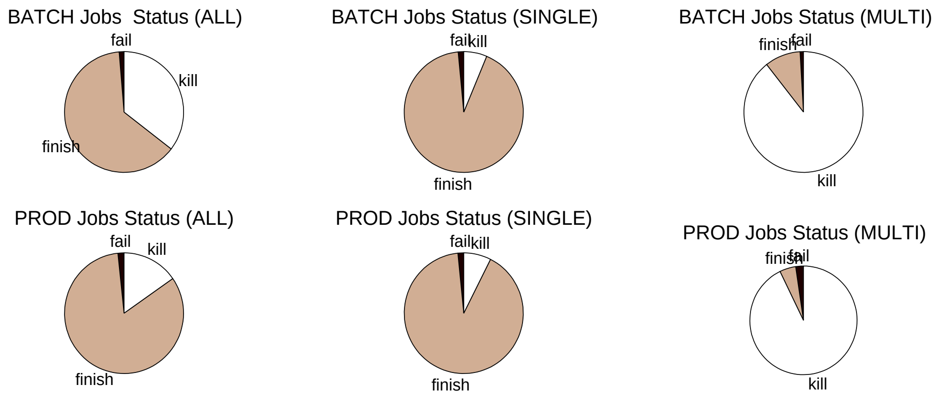


Fig. 2. Breakdown of number of jobs by their exit status in Google's cluster (left-most column: all jobs; middle column: multi-task jobs; right-most column: single-task jobs).

We observe from Figure 2 that for both production and batch jobs more than 90% of single-task jobs complete successfully, while more than 90% of multi-task jobs are eventually *killed*.

In addition to the number of job submissions in each category, we are interested in knowing the amount of time that is consumed by the jobs in the cluster, in terms of task-minutes, broken down by their exit status. Figure 3 studies the breakdown of task-minutes in single-task and multi-task jobs by job exit status. We find that for production jobs, the fraction of compute cycles consumed by jobs that are eventually killed dominates the total amount of production time in Google's cluster, regardless of whether the job is comprised of a single

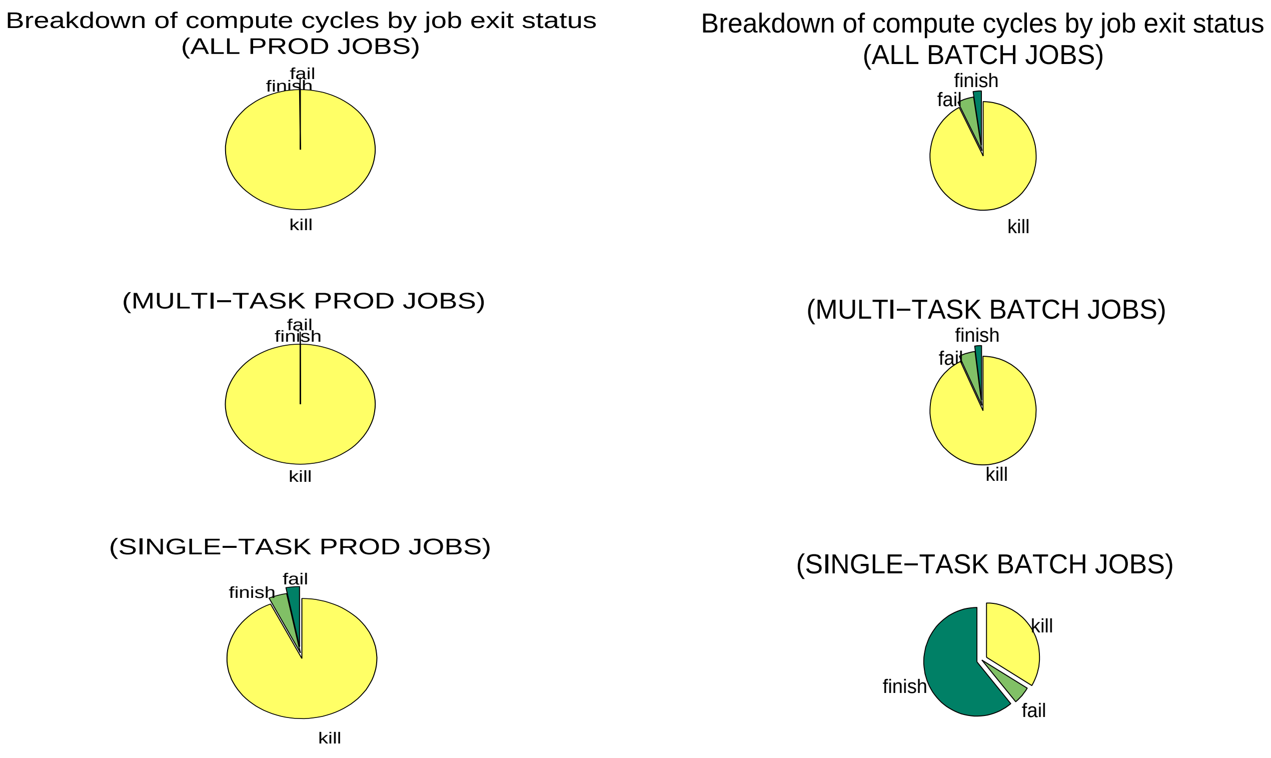


Fig. 3. Breakdown of total time consumed by jobs by exit status in Google's cluster (left column: production jobs, right-column: batch jobs).

task or multiple tasks. To further investigate why production jobs that are eventually killed dominate production time, we asked domain experts at Google about our observation and learned that some production services at Google are expected to execute long-running jobs that never actually 'complete' and are eventually terminated (i.e. 'killed') when their work is done.

We now turn to the CMU OpenCloud cluster to study the effect of number of tasks in a job. The OpenCloud workload consisted mainly of MapReduce jobs, and the number of 'mapper' tasks and 'reducer' tasks that each job spanned are made available in the traces. In Hadoop MapReduce jobs, the number of map tasks are typically driven by the size and the number of files in the input data. The ideal number of reduce tasks as recommended in Hadoop's documentation is a function of the buffer size of the output files in a job. We find that nearly 60% of the OpenCloud jobs consisted of one or zero reduce tasks, whereas more than 80% of the jobs had at least two mapper tasks. Note that the default number of reduce tasks in a job mentioned in Hadoop's official documentation is one reduce task [1].

We now study how the breakdown of a job's exit status looks like for single-task and multi-task jobs, where the number of tasks here refers to the number of *mappers*. Figure 4 compares the breakdown of the job's exit status between jobs that consist of a single mapper versus multiple mappers. We observe that unlike Google's jobs, the success rates are not different between single-task and multi-task jobs. We also observe that 50% of the time consumed by multi-task jobs (which represent 83% of all jobs submitted to the cluster) is spent on jobs that eventually *fail*. In fact, we find that 50% of the time spent on all jobs submitted to the cluster during the measurement period is spent on failed jobs.

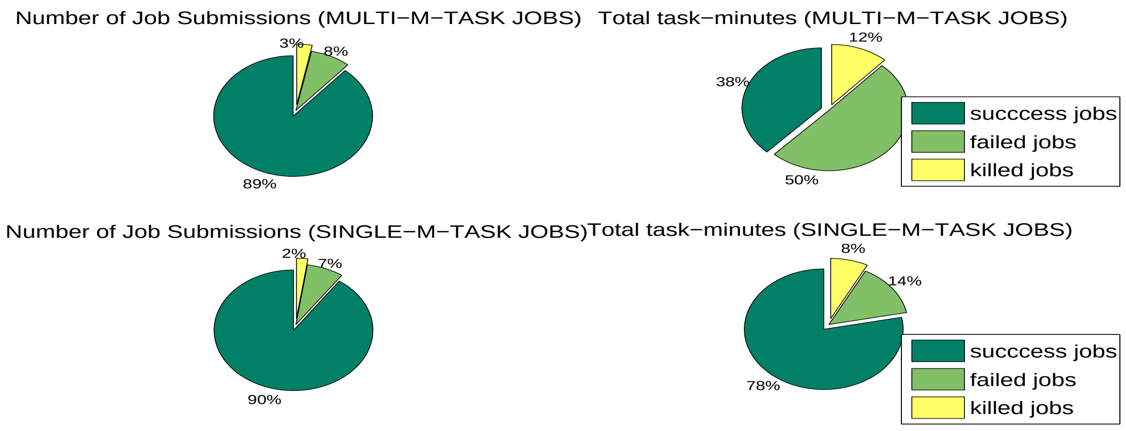


Fig. 4. Breakdown of number of jobs by their exit status in Google's cluster (left-most column: all jobs; middle column: multi-task jobs; right-most column: single-task jobs).

Finally, we examine the jobs submitted to LANL's HPC cluster. The LANL jobs do not provide information on the number of parallel *tasks* a job comprised, but the number of *nodes* that each job was assigned to run on is available and is determined by the number of processors that a job requests. Each node in LANL's cluster had four processors and jobs typically requested processors in multiples of four. Therefore, a job requesting 4 processors ran on a single node, and a job requesting more than 4 processors ran on (*#requested-processors*/4) nodes. We therefore use the number of requested processors as an indicator of the level of parallelism of a LANL job.

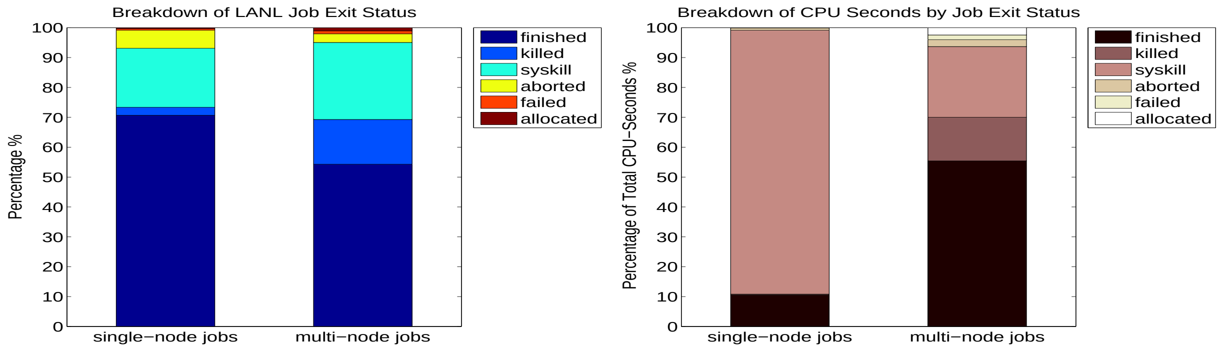


Fig. 5. Breakdown of job status for single-node and multi-node (parallel) jobs in LANL's HPC cluster.

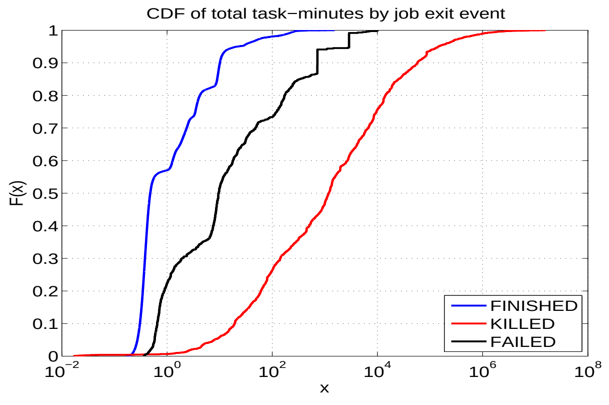
We find that 79% of the LANL jobs ran on a single node; the remaining 21% of jobs ran on varying numbers of nodes, ranging from 2 nodes and up to the entire cluster (i.e. 255 nodes). Figure 5-(left) compares the breakdown of the final status between jobs that ran on a single node and jobs that ran on multiple nodes in LANL's cluster. We observe 70% and 50% success rates in single-node and multi-node jobs, respectively. We also find that job-killing rates in particular, i.e. jobs that are either `syskilled` or `killed`, are almost doubled in multi-node jobs compared to single jobs. Figure 5-(right) compares the breakdown of the total time consumed by single- and multi-node LANL jobs broken down by job exit status. Interestingly, we find that almost 90% of the time consumed by single-node jobs is time spent on jobs that are eventually `syskilled`. Jobs that are `syskilled` are typically terminated by an administrative user either due to a problem or due to maintenance (recall Table I).

Observation: For the clusters included in our study, we find that 50% to 99% of the time spent executing parallel jobs (i.e. jobs that ran on two or more nodes), was consumed by unsuccessful jobs that end up either failing or getting aborted by the user.

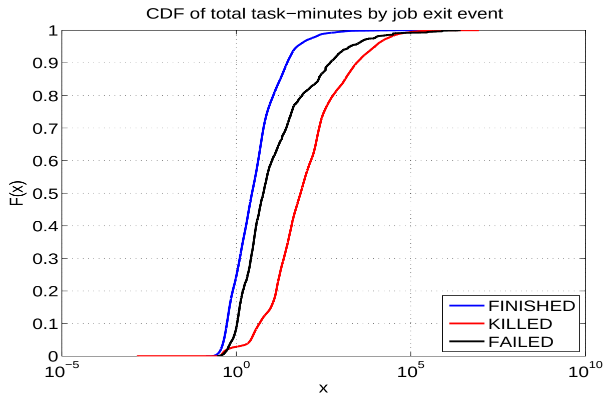
B. Job Duration

Our observations above motivated us to further study the relationship between the duration (length) of individual jobs and their exit status. The goal is to improve our understanding of what characterizes job failures or abortions in large clusters.

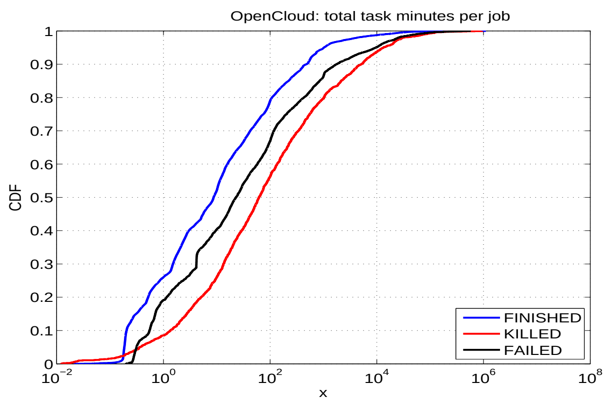
Figure 6 below studies the empirical cumulative distribution function (CDF) of job durations for all clusters in our traces. Each line in the graphs plots the CDF for jobs that share an exit status: either finish normally, fail, or get killed.



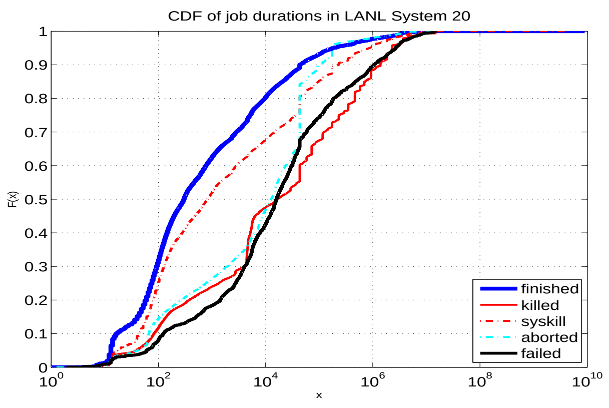
(a) Google Production Jobs.



(b) Google Batch Jobs.



(c) CMU OpenCloud Jobs.



(d) LANL Cluster jobs.

Fig. 6. Comparison of the distribution function of job durations among jobs that finish, get killed, or fail, in all clusters included in our traces.

Observation: The durations of successful jobs are consistently the *shortest* among all jobs submitted to the clusters included in our study. On the other hand, jobs that get *killed* spend the longest times running before they are terminated, with the exception of the LANL cluster where both killed and failed jobs report comparable distributions of job durations. This observation

C. Job Scheduling Attributes

We next look into the effect of different job scheduling attributes, namely the job's priority, scheduling-class, and requested resources, on the job's final status.

Job Priority: Of all the workload traces in our study, Google's data is the only one that includes information on the priority level of tasks that belong to a job (typically, all tasks in a job have the same priority). As mentioned earlier, the priority of a task is indicated by a number from 0 (lowest) to 11 (highest). Priorities 0–1 are labeled as gratis (free); priorities 2–8 are speculated to be dominated by batch jobs and MapReduce

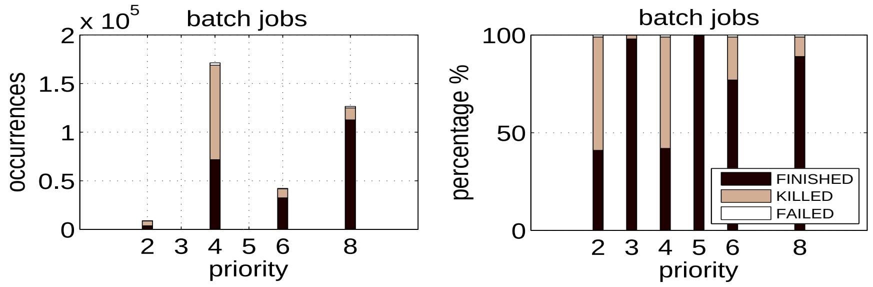


Fig. 7. Job status as a function of job priority in Google's batch jobs.

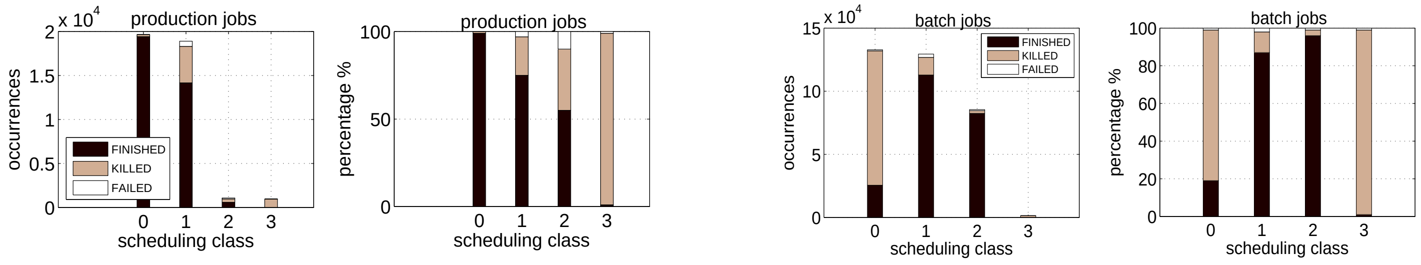


Fig. 8. Job status as a function of job scheduling-class in Google's production jobs (left) and batch jobs (right).

jobs; priority 9 is 'normal production' jobs; priority 10 represents 'monitoring' jobs; and priority 11 is labeled 'infrastructure'.

When assigning tasks to machines, tasks with higher priorities are favoured in resources over tasks with lower priorities. Additionally, the scheduler in Google's cluster is designed such that over-committing resources on a machine is permissible. Therefore, in the case when there are not enough resources to satisfy the requests of all tasks running on a machine, tasks with lower priorities may be killed [10].

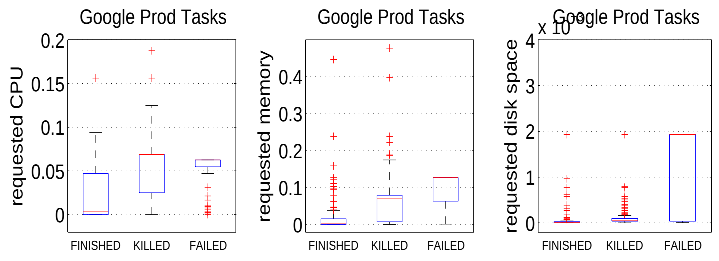
In our analysis so far we have been focusing on studying and comparing jobs from two priority groups: normal production jobs (priority 9) and batch jobs (priorities 2–8). We now take a closer look at the range of different priority levels within *batch* jobs. Figure 7 plots the number of scheduled jobs as a function of job priority, while showing the breakdown of job statuses; the left plot shows the absolute counts of jobs and the right plots shows the breakdown in terms of percentages. We find that the top priority levels assigned to batch jobs by users were 4 (49% of batch jobs), 8 (36%) and 6 (12%), and that the success rates in those three groups are correlated positively with the priority level (i.e. the higher the priority, the higher the success rate).

Job Scheduling Class: In Google's cluster, jobs have a scheduling class that determines how latency-sensitive a job is, with values ranging from 0 (least sensitive), to 3 (most sensitive). While a task priority determines if it is scheduled on a machine or not, the scheduling class is used *locally* by the machine to implement machine-local policies for accessing its resources.

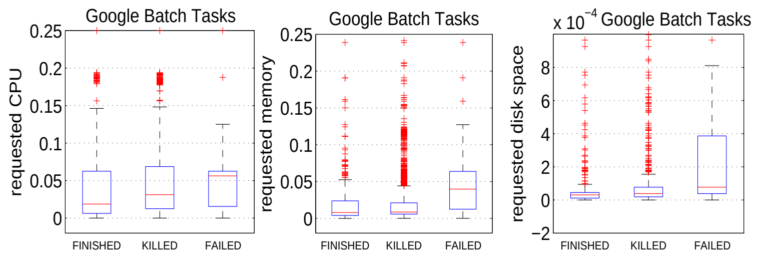
Figure 8 shows the breakdown of job status as a function of scheduling class for both production jobs and batch jobs in Google's cluster. We find that scheduling classes 0 and 1 are the two most commonly chosen classes by users for both production and batch jobs. In production jobs, the higher the scheduling class of a job the lower its success chance; an opposite trend is observed in batch jobs, however, with the exception of scheduling class 3. In both production and batch jobs, scheduling class 3 is the least chosen by users and approximately 99% of jobs under this class are eventually killed.

Observation: In the Google cluster in our dataset, the priority level of a batch job was positively correlated with the job's success chance. Weaker trends were observed between the scheduling class of a job and its exit status, when considering both batch jobs and production jobs.

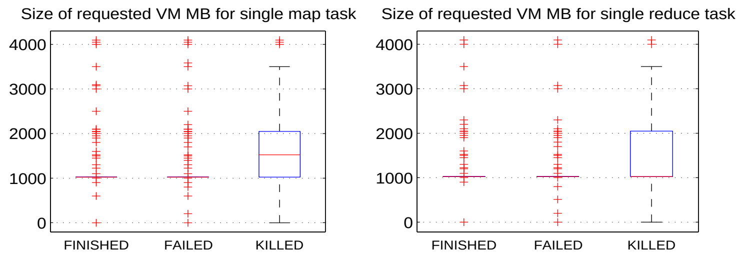
Requested Resources: We now study the relationship between the amount of resources requested by tasks in a job the final status of the job. In the LANL cluster, we only have data on the number of requested processors per job. In the Google cluster and the OpenCloud cluster, more detailed information is available. In Google, tasks are submitted with values for requested CPU, memory, or disk space, where these values represent the



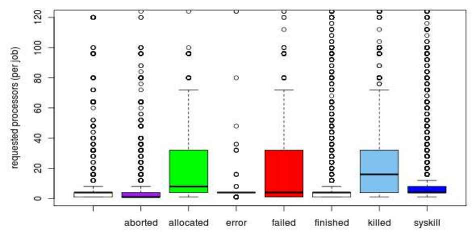
(a) Google Production Jobs.



(b) Google Batch Jobs.



(c) CMU OpenCloud Jobs.



(d) LANL Jobs.

Fig. 9. Requested resources by tasks in a job versus the exit status of a job in the Google cluster and CMU's OpenCloud cluster.

maximum amount of resources a task is allowed to consume on a machine. However, tasks are sometimes permitted to use more than what they requested if resources are available; e.g. tasks may use free CPU cycles on a machine [10].

In the OpenCloud cluster, job configuration parameters include the following fields for requested resources (more detailed definitions are found in Hadoop's documentation [1]):

- `mapred.job.map.memory.mb`: The size of virtual memory for a single map task.
- `mapred.job.reduce.memory.mb`: The size of virtual memory for a single reduce task.

Figure 9 plots the distribution of the average requested resources by tasks in a job, for jobs that finish, fail or get killed, separately. We use boxplots [1] to examine and compare the distribution of requested resources.

The first thing we observe from the graphs is that resource requests varied between successful and unsuccessful job runs in all the clusters in our data. In Google's cluster, production jobs that *failed* had a significantly higher median value for requested CPU, memory, or disk space than successful jobs; production jobs that were *killed* had higher CPU and memory requests in particular (see Figure 9-(a)). For Google's batch jobs, we observe higher median values for requested memory and disk space in jobs that *failed* (see Figure 9-(b)).

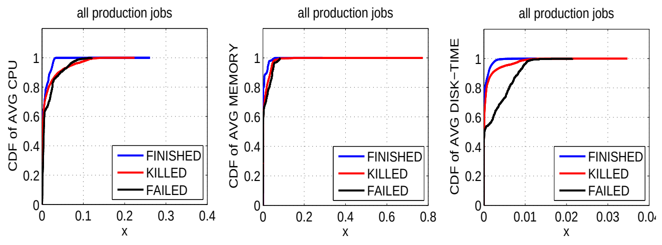
In the OpenCloud cluster, we find that jobs that were eventually *killed* report significantly higher requests for virtual memory than successful jobs or even jobs that failed, for both mapper and reducer tasks (see Figure 9-(c)). In the LANL cluster, we observe significantly higher distributions of the number of requested processors by a job in jobs that get *killed*, compared to successful jobs.

Observation: The observed correlations between the amount of requested resources by a job and its unsuccessful termination in the clusters that are included in our study suggest that knowledge of how much CPU, memory, or disk space a job requests can be used as predictors of its final status. We investigate the potential of using requested resources as predictors of job failures in the next Section, Section IV.

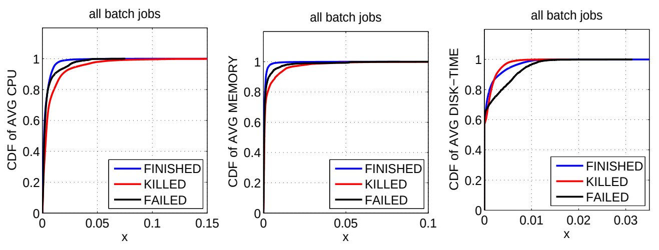
D. Job Resource Usage

After examining how a job's configuration parameters such as its degree of parallelism, scheduling constraints or requested resources correlate with the job's exit status, we now turn our attention to the actual resource consumption of a job. Our goal is to study how the way a job utilizes resources in a cluster affects its final status.

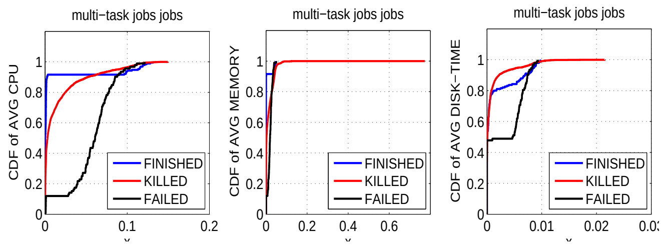
[1]Recall that in a box plot the bottom and top of the box are always the 25th and 75th percentile, respectively, and the band near the middle of the box is always the 50th percentile (the median).



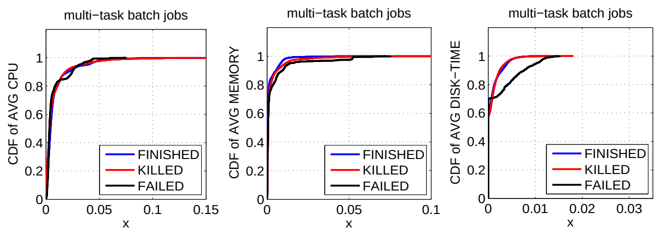
(a) Google Production Jobs.



(b) Google Batch Jobs.



(c) Google Production Jobs (Multi-Task).



(d) Google Batch Jobs (Multi-Task).

Fig. 10. The average resource utilization in Google's jobs, for jobs that finish successfully, fail, or get killed. Figures (a,b) include all production and batch jobs; figures (c,d) include multi-task jobs only.

The only dataset in our traces that allows us to explore this question is the data from Google's cluster where task usage information is made available. In particular, for each task running on a machine in the cluster, there is data on CPU, memory, and disk usage, aggregated over 5-minute intervals. All values of usage data are normalized by the maximum value reported in each category.

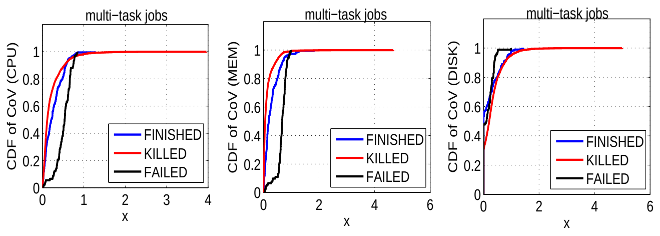
The graphs in Figure 10-(a,b) show the distribution of the average CPU, memory, and disk I/O consumption for all production jobs and batch jobs submitted to the Google cluster, plotted separately for different job exit statuses. We find that jobs that *fail* reported the highest disk I/O consumption, on average, both in production jobs and batch jobs. More precisely, the average I/O usage by a failed job was 10X and 2X times higher than that of a successful job in production jobs and batch jobs, respectively.

When repeating this analysis for *multi-task* jobs only in Figure 10-(c,d), we find that multi-task production jobs in particular reported significantly higher values for both CPU and disk I/O consumption, compared to successful jobs. For example, the average CPU usage of a failed multi-task production job was 284X times higher than that of a successful job. Jobs that were killed, on the other hand, reported the highest values for *memory* usage in particular; for example, the average memory usage of a killed job was 10X and 4X higher than a successful job in production jobs and batch jobs, respectively.

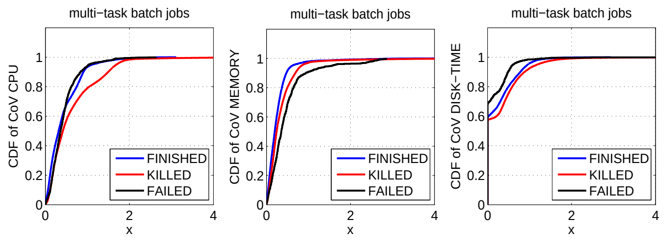
Variability in job resource usage: In addition to studying and comparing the distribution of the average resource consumption of jobs, we also ask the question of how the *variability* in usage between tasks that belong to the same job compares across successful and unsuccessful jobs. Figure 11 in the next page plots the distribution of the coefficient of variation (CoV) in resource usage between tasks in a job. Note that we only include multi-task jobs in this analysis.

Interestingly, we find that failed jobs which had the highest average in I/O usage, report the lowest variability in I/O usage between tasks working on the same job. Similarly, killed jobs which reported the highest average in memory consumption (especially in production jobs), experienced the lowest variability in memory usage between tasks in a job. We also find that failed jobs had the highest variability in the amount of memory used by tasks in a job, in both production jobs and batch jobs.

Estimation of requested resources: So far we have studied the correlation between the *requested* resources by a job, and the actual *usage* of a job, separately. Another question one may ask is whether the over- or under-estimation of requested resources by a job affected the final status of the job. To study this question, Figure 12 plots the ratio between a job's actual average (and maximum) resource consumption values and the job's requested values, for CPU and memory. The dashed green line indicates a ratio of 1, therefore any value

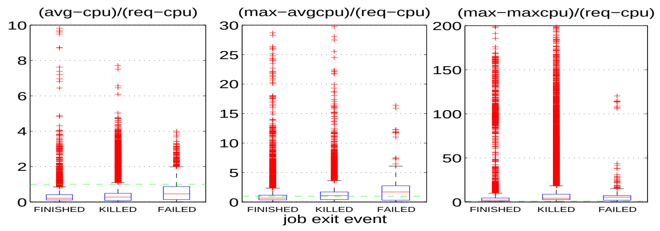


(a) Google Production Jobs (Multi-Task).

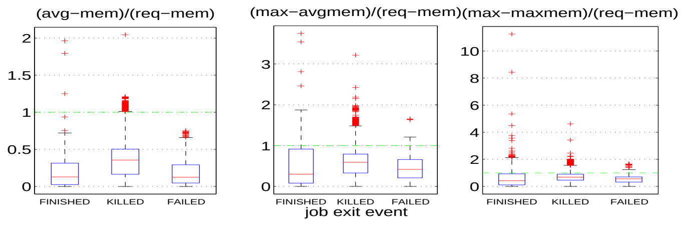


(b) Google Batch Jobs (Multi-Task).

Fig. 11. The variability in resource utilization between tasks in a multi-task job, for jobs that finish successfully, fail, or get killed in Google's cluster.



(a) CPU usage.



(b) Memory usage.

Fig. 12. The ratio between requested resources by jobs and the actual usage of jobs.

above the dashed line indicates an under-estimation of resources while values below the line indicate an over-estimation. (Recall that the values for 'requested' resources in Google's cluster are defined as the maximum anticipated usage values by the tasks belonging to a job [10].)

We observe that failed jobs had the highest portion of jobs that reported a maximum CPU usage value exceeding their requested CPU capacity. Jobs that were killed, on the other hand, reported the highest median of the ratio between consumed memory and requested memory; i.e. these jobs used more memory than what they had initially requested.

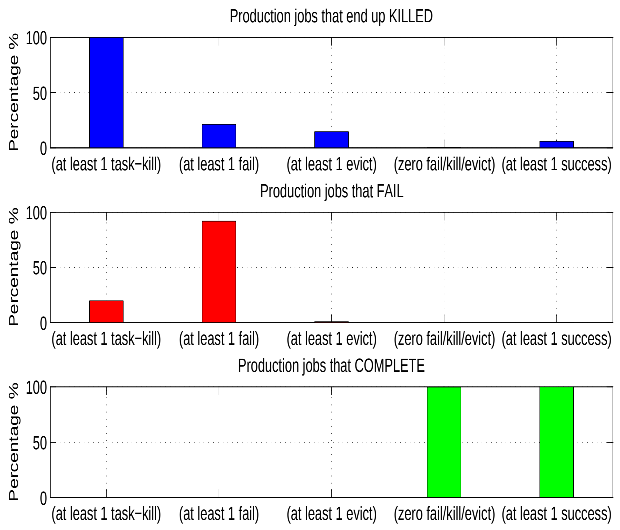
Observation: For the Google cluster in our data, the resource utilization of jobs in terms of CPU, memory and I/O consumption, varied between jobs that completed successfully, jobs that failed, and jobs that were killed. This observation suggests that how jobs access and use resources in large clusters can be used to characterize job reliability and to potentially predict the exit status of a job.

E. Task Reliability

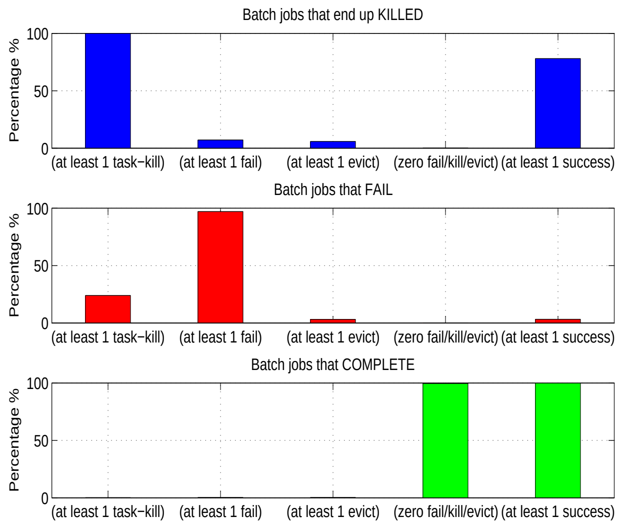
In this subsection we turn our attention to the *task* level in large-scale jobs. In both Google and OpenCloud clusters, jobs consisted of one or more tasks. The cluster scheduler assigns tasks to machines, where a task can be running on a single machine at any point in time. Tasks can complete successfully, fail, get killed, or get evicted (in Google, task evictions happen due to resource overcommitment or when higher priority tasks arrive at a machine and the task demands exceed machine capacity [10]). Our goal is to understand how the completion of a job is affected by the reliability of its tasks. We study different task fault-tolerance configuration parameters, while focusing on task resubmissions.

We begin by asking the question of how the statuses of task attempts in a job relate to the final status of the job. Figure 13 plots the fraction of jobs in Google's cluster that experienced at least one task failure, task kill, or task eviction, for jobs that end up getting killed, failing, or completing successfully.

Based on Figure 13, we make the following observations about jobs and tasks in Google's cluster:
• We observe a strong association between having at least one task failure and the entire job eventually failing. Similarly, a single task kill event is highly associated with a job getting killed.
• It is extremely rare for jobs that complete successfully to experience any unsuccessful task attempts (see last row of graphs).
• Jobs that end up being killed can have successful task attempts. For example, 78% of killed batch jobs had at least one task complete successfully.



(a) Google Production Jobs.



(b) Google Batch Jobs.

Fig. 13. The relationship between task exit status and job exit status in Google's cluster.

• Task evictions show the strongest correlation with job *kills*. Further investigation shows that 98% and 90% of production jobs and batch jobs, respectively, which experienced at least one task evict, were eventually killed.

We repeat this analysis for the CMU OpenCloud cluster in Figure 14 below. We find that a higher fraction of successful jobs experienced failed or killed task attempts, compared to Google's jobs (note that the OpenCloud traces had no records of task 'evicts'). We also find that the status of (unsuccessful) mapper tasks in a job were more indicative of the job's final status than reducer tasks.

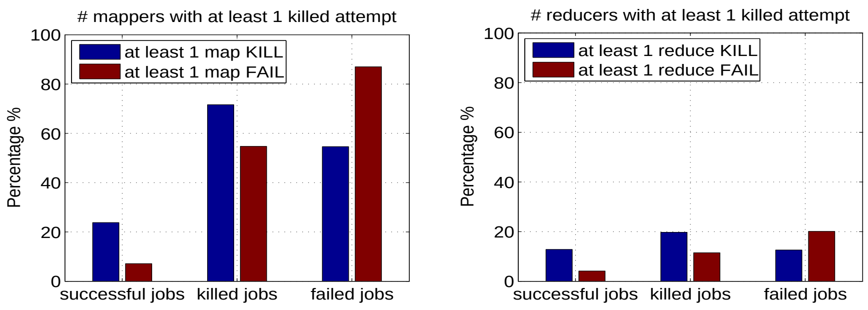
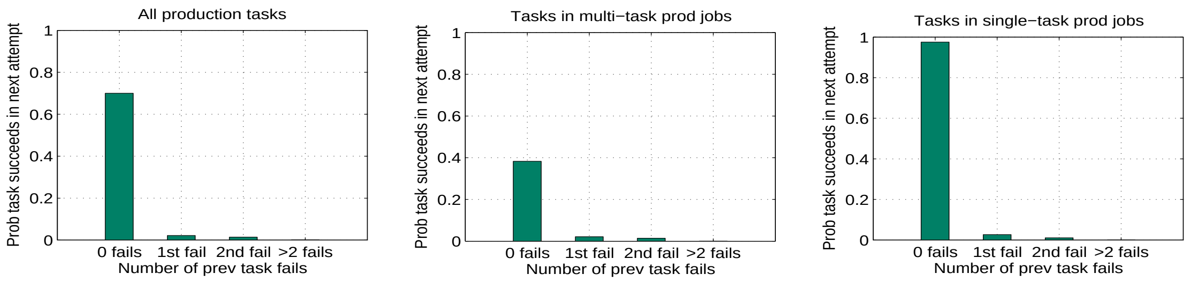


Fig. 14. The relationship between task exit status and job exit status in the OpenCloud cluster.

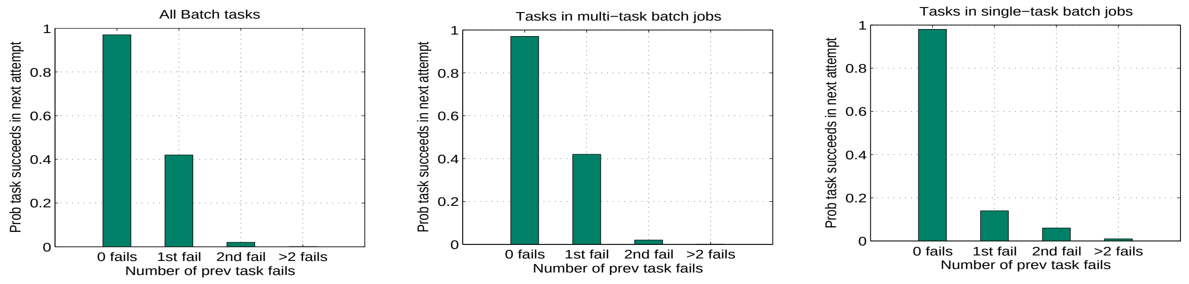
Going back to the Google cluster, our observation that a single task failure was highly predictive of a job's unsuccessful termination motivated us to take a closer look into the nature of task failures. Note that we focus on task *failures* in particular and not task 'kills' which we exclude from this part of our analysis, since tasks are typically killed in Google's cluster once their job is killed (either by the user or by another dependent job [10]). Task failures, on the other hand, are usually the result of a software crash, and when a task fails it is automatically resubmitted to the cluster [7].

We next investigate the effectiveness of task resubmissions due to failures by studying the likelihood of a task completing successfully, after making previous failed attempts.

Figure 15 plots the probability of a task *succeeding* in the next attempt as a function of the number of past *failed* attempts it had made in Google's cluster. The left-most graphs show the results when taking all jobs into consideration; the middle plots are for multi-task jobs only; and the right-most plots are for single-task jobs.



(a) Google Production Jobs.



(b) Google Batch Jobs.

Fig. 15. The probability that a task succeeds in an attempt as a function of the number of past failed attempts in the Google cluster.

The first observation we make from the figure is that when a task has no history of failing, its success chance is 70% if it is a production task and 98% if it is a batch task. When considering tasks that belong to multi-task jobs only, a task's chance in succeeding without any past failures drops in production jobs to 38%. Tasks in single-task jobs, however, have a 99% probability of succeeding in both batch and production jobs, given that no past attempts have failed.

If one task attempt *fails*, the task's chance of succeeding in the next attempt drops dramatically to 2% only in production tasks, and to 42% in batch tasks. If a task fails more than twice, its success chance in subsequent attempts becomes negligible, regardless of its priority level.

Observation: In the Google cluster, the chance of a task attempt completing successfully drops significantly once the task experiences at least one attempt failure. This observation suggests that terminating a job after the first task failure takes place can potentially save significant resources in the cluster, instead of continuing to run a job with low chances of succeeding.

IV. Job Failure Prediction

So far in this work we have studied the impact of various factors on job reliability in large-scale clusters. Our goal in this section is to utilize the knowledge obtained from our trace-based analysis in predicting job failure events in large clusters. We apply our classification and prediction techniques to the traces made available by Google and by CMU's OpenCloud cluster, since these two traces contain more detailed information on jobs.

We use two different classification techniques to classify jobs according to their final status in the cluster, i.e. finished, failed, or got killed: Multinomial Logistic Regression (MLR), and Classification and Regression Trees (CART). We apply our classification and prediction techniques on each data source separately, and since some workload traces contain additional data fields about the jobs, our set of predictors can differ from one trace to another.

1- Google Cluster: We begin by looking at Google's job traces. Our explanatory (predictor) input variables extracted from the trace, which we feed our classifiers in order to learn and predict a job's final status, are summarized in Table III below:

We apply the two classifiers, MLR and CART, for Google's production jobs and batch jobs, separately. For each dataset, we split the data randomly such that 70% of the rows are used for training the MLR and CART

Trace	Variable	Category	Description
	`job_status`	Response	The outcome we are interested in predicting. A job's status can either be `FINISHED`, `FAILED`, or `KILLED`.
Input Variables (Predictors)			
Google	`userID`	Job Config	The ID of the user that submitted this job (either a Google engineer or a service).
	`logical_job_name`	Job Config	The name of the program (application) that this job is running.
	`scheduling_class`	Job Config	Each job has a scheduling class that drives machine local policies for allocating resources.
	`priority`	Job Config	Job and task priorities determine if they are given preferences for resources.
	`num_tasks`	Job Config	Each job spans one or more tasks. This determines the degree of parallelization of a job.
	`different_machines_rest`	Job Config	This flag, when enabled, means that all tasks in a job must be scheduled on different physical machines.
	`requested_cpu`	Job Config	The amount of requested CPU capacity by a task in a job.
	`requested_memory`	Job Config	The amount of memory requested by a task in a job.
	`requested_disk`	Job Config	The amount of disk space requested by a task in a job.
	`num_completed_tasks`	Job Counters	Number of tasks in a job that completed successfully.
	`num_failed_tasks`	Job Counters	Number of tasks in a job that experienced one or more failed attempts during the job's lifetime.
	`num_killed_tasks`	Job Counters	Number of tasks in a job that experienced one or more killed attempts during the job's lifetime.
	`num_evicted_tasks`	Job Counters	Number of tasks in a job that experienced one or more evicts during the job's lifetime.
	`num_bad_tasks`	Job Counters	Number of tasks in a job that experienced one or more failed/killed/evicted attempts during the job's lifetime.
	`job_duration`	Job Counters	The total amount of task-minutes spent by a job.
	`avg_cpu_usage`	Job Usage	The average amount of CPU used by a job's tasks.
	`avg_memory_usage`	Job Usage	The average amount of memory used by a job's tasks.
	`avg_IO_usage`	Job Usage	The average amount of I/O used by a job's tasks.
	`CoV_cpu_usage`	Job Usage	The variability in CPU usage among tasks belonging to the same job.
	`CoV_memory_usage`	Job Usage	The variability in memory usage among tasks belonging to the same job.
	`CoV_IO_usage`	Job Usage	The variability in I/O usage among tasks belonging to the same job.

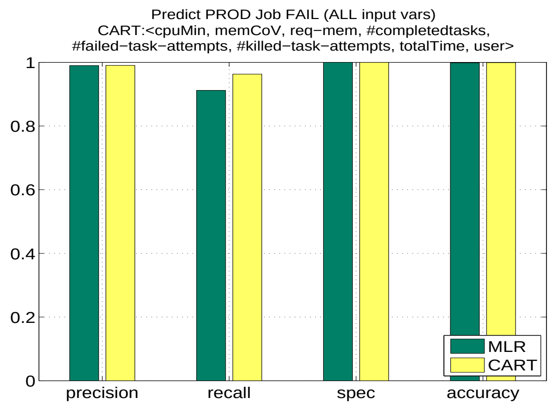
TABLE III
SUMMARY OF JOB STATUS CLASSIFICATION AND PREDICTION VARIABLES FOR THE GOOGLE CLUSTER.

models, and 30% are used for testing. We use standard metrics to evaluate the quality of our predictors:
- Precision: the percentage of predicted job failure events that are true.
- Recall: the percentage of actual job failure events that were successfully predicted.
- Specificity: the percentage of negative job failures (i.e. job successes or kills) that were correctly labeled.
- Accuracy: the percentage of predictions that are correct.

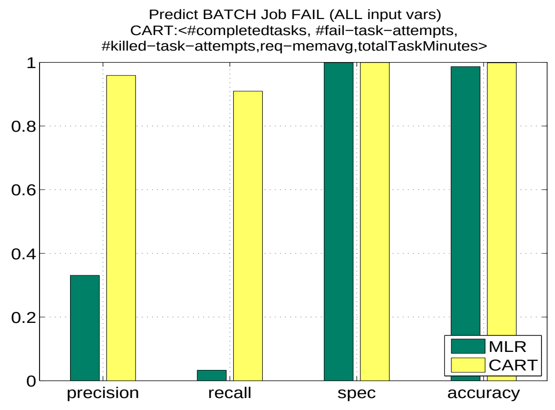
a) Predicting a job's status based on all input variables. We begin by classifying a job's final status while assuming knowledge of all the attributes and counters related to a job. Figure 16 plots the results of our classification models when using MLR and CART techniques, and when using the data for *all* the variables described in Table III as predictors. While, realistically, obtaining knowledge about some of these fields cannot be done in practice until a job stops running (e.g. the total number of completed tasks, etc), it is still useful to run and test our models on all the variables in order to develop a better understanding of which job attributes are most relevant to a job's final status. This can be done particularly using CART, where the output of running the model on the data produces a list of the attributes that ended up being used by the classifier to construct the tree. Note that we included these lists of 'relevant attributes' to the graph titles in Figure 16.

We make several observations from Figure 16. First, we find that CART performs better than MLR in predicting job failures, with approximately 99% precision and 90–95% recall. Next, we observe that most of the attributes used by CART to classify a job successfully were in common between batch jobs and production jobs; in particular, we find that CART used the following fields: the amount of memory requested by a job, the number of tasks in a job with successful/unsuccessful attempts, and the total time spent by a job in the cluster. The two additional attributes used by production jobs were user ID and the variability in memory usage over time.

The above results point us to which attributes are more significant to a job's terminal status than others



(a) Google Production Jobs.



(b) Google Batch Jobs.

Fig. 16. Evaluation of predicting job failures in the Google cluster when using *all* input data as predictors.

whenever all job attributes and counters are considered. Next, instead of assuming knowledge of all attributes, we ask the question of whether we can predict a job's status based on the job's configuration parameters and resource requests that are known to the user before the job is executed.

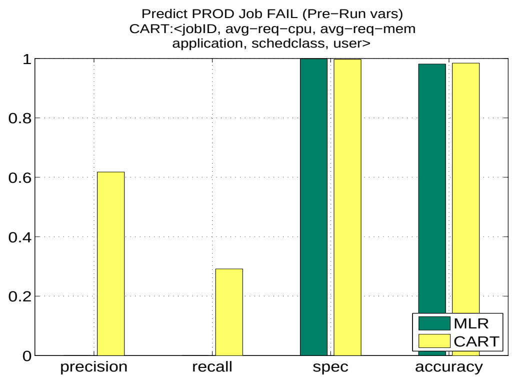
b) Predicting a job's status based on config parameters. We now consider the attributes under the category 'Job Config' in Table III which are known to the user prior to a job's execution: the user and application IDs, the job's priority and scheduling class, the number of tasks in the job, the amount of requested resources by a job, and whether or not there is an anti-affinity constraint for the job's tasks.

Figure 17 shows the prediction results when using these pre-run job attributes and constraints. We find that using configuration parameters alone dropped our recall percentage significantly to 20–30%; i.e. 20% to 30% of failed jobs were correctly predicted by our CART classifier. Precision remained significant (99%) for batch jobs but dropped to 60% in production jobs. Interestingly, we find that for batch jobs this high precision was achieved while using only one attribute for classifying jobs: the amount of requested disk space. The attributes used for classifying production jobs, on the other hand, were different and included the requested CPU and memory capacity, the application name, the scheduling class of the job, and the user ID.

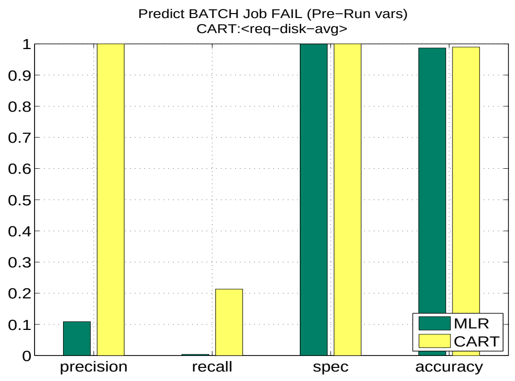
We next investigate whether these predictions can be improved if we consider the status of *tasks* that make failed attempts during the lifetime of a job.

c) Predicting a job's status based on config parameters and a task-failure flag. One of the observations we make in Section III-E is that a single task failure event is a strong indicator of both the task and the job eventually failing. This observation motivated us to study if we can predict a job's final status accurately as soon as one of its tasks makes a failed attempt to execute. To study this question, we include in our input data to the classifiers a single flag that is set to true if at least one task attempt failed (`didTaskFail`).

We plot the results of our predictions when using this flag (in addition to the pre-run job configuration attributes), in Figure 18. We observe that including the `didTaskFail` flag improved our prediction results significantly, compared to when only config parameters were considered. We find that for both production and batch jobs more than 80% of job failures are predicted successfully, and the precision of our predictor exceeds 85%. We conclude that a single task failure attempt is a strong predictor of a job's final status and can be used in conjunction with job configuration parameters to predict a job's failure probability with high accuracy.

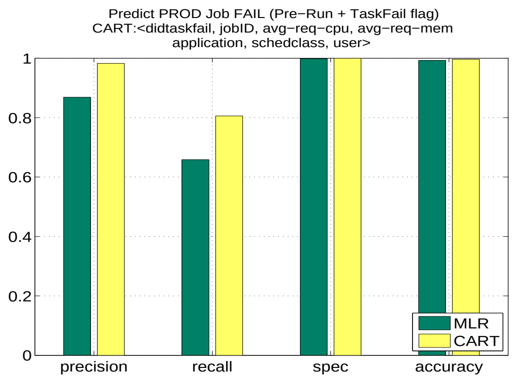


(a) Google Production Jobs.

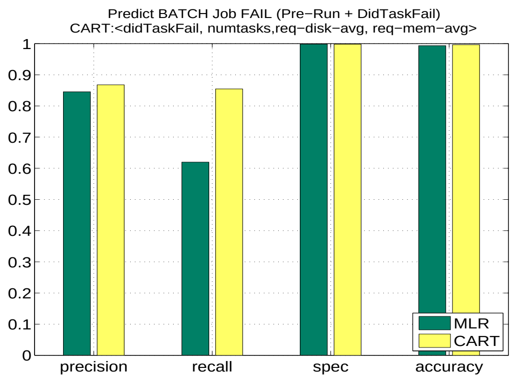


(b) Google Batch Jobs.

Fig. 17. Evaluation of predicting job failures in the Google cluster when using job configuration and constraints attributes only as predictors.



(a) Google Production Jobs.



(b) Google Batch Jobs.

Fig. 18. Evaluation of predicting job failures in the Google cluster when adding a flag for at least one failed task attempt as a predictor in addition to job config parameters.

2- CMU OpenCloud cluster: We now repeat our job failure classification and prediction analysis on the workload traces provided by CMU's OpenCloud cluster. Table IV in the next page summarizes the job attributes available for the OpenCloud cluster jobs.

Similar to our job failure prediction analysis for the Google jobs, we ran our MLR and CART classifiers on the OpenCloud data to classify and predict job failure events, first while including all the possible input variables describing a job, then by including variables that are known only at job configuration time (i.e. before job execution starts), and finally, we tested our predictors when considering a `didTaskFail` flag as a predictor variable in addition to the job config attributes. Figure 19 shows the results for our different prediction attempts.

We observe from the graphs in Figure 19 that once again CART performs better in classifying job failures than MLR. When including all job attributes, CART is able to successfully classify 100% of all the job failures that happened in the OpenCloud cluster, with 80% precision (i.e. 20% of the jobs classified as failures were false negatives). The attributes used by CART were the total counts of failed task attempts made by mappers and by reducers, the number of reducers that completed successfully, and the number of mappers and reducers with at least one failed attempt.

When considering only job config parameters, the recall percentage of job failures drops significantly to 12%, but when adding the `didTaskFail` flag indicating that at least one task attempt has failed in a job it increases to 55%, and the amount of requested memory for reducer tasks is considered by the classification tree.

Observation: For the clusters in our dataset, job status classification and prediction can be achieved using classification and regression trees (CART). Knowledge of at least one failed task attempt, in addition to job config parameters, was a strong predictor of a job failing. Additionally, the amount of requested resources by a job, especially memory capacity, was used by our classifiers when predicting job failure events across different clusters.

Trace	Variable	Category	Description
	`job_status`	Response	The outcome we are interested in predicting. A job's status can either be `FINISHED`, `FAILED`, or `KILLED`.
		Input Variables (Predictors)	
OpenCloud	`userID`	Job Config	The ID of the user that submitted this job (CMU researcher).
	`JobTrackerID`	Job Config	This is the ID generated when a Hadoop JobTracker object is booted. A JobTracker launches jobs and assigns IDs to them.
	`JobID`	Job Config	The combination of a JobTrackerID and a JobID is a unique identifier of a job.
	`num_maps`	Job Config	The number of mapper tasks in a job.
	`num_reduces`	Job Config	The number of reducer tasks in a job.
	`max_attempts_map`	Job Config	The maximum number of attempts a map task can make before the framework gives up on it.
	`max_attempts_reduce`	Job Config	The maximum number of attempts a reduce task can make before the framework gives up on it.
	`skip_attempts`	Job Config	The number of task attempts after which 'skip' mode will be kicked off. Skip mode means a task reports the records it will process next so that on failures the framework would know which data records are possibly bad records.
	`map_speculate`	Job Config	If true, then multiple instances of some map tasks may be executed in parallel.
	`red_speculate`	Job Config	If true, then multiple instances of some reduce tasks may be executed in parallel.
	`requested_memory_map`	Job Config	The size of VM requested for a single map task.
	`requested_memory_red`	Job Config	The size of VM requested for a single reduce task.
	`finMaps`	Job Stats	Number of completed map tasks in the job.
	`finReduces`	Job Stats	Number of completed reduce tasks in the job.
	`failMaps`	Job Stats	Total number of unsuccessful attempts by mapper tasks in the job.
	`failReduces`	Job Stats	Total number of unsuccessful attempts by reducer tasks in the job.
	`maps_atleast_1fail`	Job Stats	Number of map tasks with at least one failed attempt.
	`red_atleast_1fail`	Job Stats	Number of reduce tasks with at least one failed attempt.
	`maps_atleast_1kill`	Job Stats	Number of map tasks with at least one killed attempt.
	`red_atleast_1kill`	Job Stats	Number of reduce tasks with at least one killed attempt.
	`job_duration`	Job Stats	The total amount of task-minutes spent by a job.
	`submitTime`	Job Stats	The timestamp for the time a job was submitted to the cluster.
	`launchTime`	Job Stats	The timestamp for the time a job was launched.
	`finishTime`	Job Stats	The timestamp for the time a job finished running.

TABLE IV
SUMMARY OF JOB STATUS CLASSIFICATION AND PREDICTION VARIABLES FOR THE CMU OPENCLOUD CLUSTER.

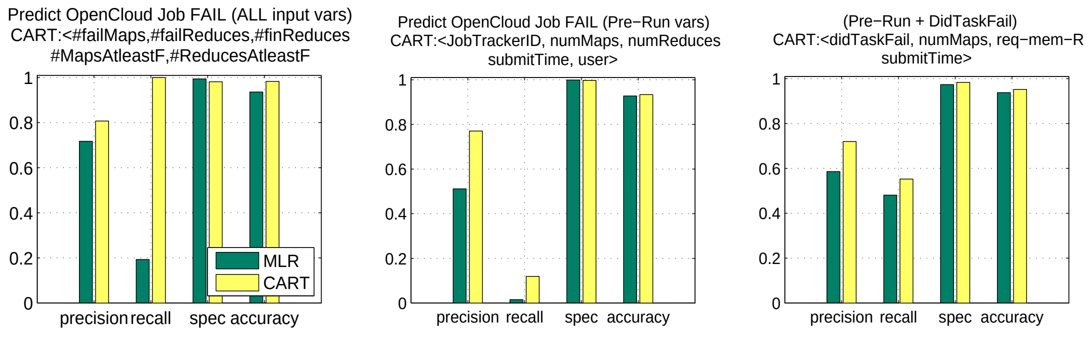


Fig. 19. Evaluation of predicting job failures in the Google cluster when using *all* input data as predictors.

V. Conclusion

In this technical report, we analyzed workload traces collected at multiple large-scale clusters to study the question of how reliable are jobs that are executed on large, parallel clusters, and what makes these jobs prone to failures (i.e. crashes) or abortions.

We first characterized unsuccessful jobs in our dataset by studying how different job attributes are distributed in the traces across jobs that finish successfully, fail, or get killed. We investigated the effect of various factors on a job's terminal status, including the degree of parallelization of a job (i.e. the number of parallel tasks the job spans), job duration, job scheduling attributes such as priority and scheduling class, resource constraints and requests, resource utilization (e.g. CPU, memory, and I/O usage), and task reliability. Our results show strong correlations between the number of parallel tasks a job spans and the amount of resources requested or used by the job's tasks.

We then utilized this knowledge in building a prediction model using classification and regression trees with the goal of predicting job failure events. Our results show that job failures can be predicted with high accuracy just by knowing whether a single task attempt failed or not, in conjunction with knowledge of job configuration attributes (e.g. the amount of memory requested by a job or the number of tasks the job spans, etc).

References

[1] Apache's Hadoop DFS. http://hadoop.apache.org/.

[2] CMU Parallel Data Lab Project. http://www.pdl.cmu.edu/HLA/index.shtml.

[3] OpenCloud Hadoop cluster trace: format and schema. http://ftp.pdl.cmu.edu/pub/datasets/hla/dataset.html.

[4] Operational Data to Support and Enable Computer Science Research, Los Alamos National Laboratory. http://institute.lanl.gov/data/fdata/.

[5] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1), 2014.

[6] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 78:1–78:12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[7] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012. ACM.

[8] B. Schroeder and G. A. Gibson. Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you? In *Proc. of the 5th USENIX conference on File and Storage Technologies*, Berkeley, CA, USA, 2007. USENIX Association.

[9] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. Debardeleben, P. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. S. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen. Addressing failures in exascale computing. 2013.

[10] J. Wilkes. More Google cluster data. Google research blog, Nov. 2011. Posted at URL http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.