Università della Svizzera italiana

Faculty of Informatics

Bachelor Thesis

May 24, 2021

Understanding and Comparing Unsuccessful Executions in Large Datacenters

Claudio Maggioni

Abstract

The project aims at comparing two different traces coming from large datacenters, focusing in particular on unsuccessful executions of jobs and tasks submitted by users. The objective of this project is to compare the resource waste caused by unsuccessful executions, their impact on application performance, and their root causes. We will show the strong negative impact on CPU and RAM usage and on task slowdown. We will analyze patterns of unsuccessful jobs and tasks, particularly focusing on their interdependency. Moreover, we will uncover their root causes by inspecting key workload and system attributes such asmachine locality and concurrency level.

Advisor
Prof. Walter Binder
Co-advisor
Dr. Andrea Rosá

Advisor's approval (Prof. Walter Binder, Dr. Andrea Rosá): Date:

Contents

1 Introduction

In today's world there is an ever growing demand for efficient, large scale computations. The rising trend of "big data" put the need for efficient management of large scaled parallelized computing at an all time high. This fact also increases the demand for research in the field of distributed systems, in particular in how to schedule computations effectively, avoid wasting resources and avoid failures.

In 2011 Google released a month long data trace of their own cluster management system[1] *Borg*, containing a lot of data regarding scheduling, priority management, and failures of a real production workload. This data was 2009 This data was the foundation of the 2015 Rosá et al. paper *Understanding the Dark Side of Big Data Clusters: An Analysis beyond Failures*[2], which in its many conclusions highlighted the need for better cluster management highlighting the high amount of failures found in the traces.

In 2019 Google released an updated version of the *Borg* cluster traces[3], not only containing data from a far bigger workload due to improvements in computational technology, but also providing data from 8 different *Borg* cells from datacenters located all over the world. These new traces are therefore about 100 times larger than the old traces, weighing in terms of storage spaces approximately 8TiB (when compressed and stored in JSONL format)[4], requiring a considerable amount of computational power to analyze them and the implementation of special data engineering techniques for analysis of the data.

This project aims to repeat the analysis performed in 2015 to highlight similarities and differences in workload this decade brought, and expanding the old analysis to understand even better the causes of failures and how to prevent them. Additionally, this report will provide an overview on the data engineering techniques used to perform the queries and analyses on the 2019 traces.

2 Background information

Borg is Google's own cluster management software able to run thousands of different jobs. Among the various cluster management services it provides, the main ones are: job queuing, scheduling, allocation, and deallocation due to higher priority computations.

The core structure of Borg is a cell, a set of machines usually all within the same cluster, whose work is allocated by the same cluster-management system and hence a cell is handled as a unit. Each cell may run large computational workload that is submitted to Borg. Such workload is called "job", which outlines the computations that a user wants to run and is made up of several "tasks". A task is an executable program, consisting of multiple processes, which has to be run on a single machine. Those tasks may be ran sequentially or in parallel, and the condition for a job's successful termination is nontrivial.

2.1 Traces

The data relative to the events happening while Borg cell processes the workload is then encoded and stored as rows of several tables that make up a single usage trace. Such data comes from the information obtained by the cell's management system and single machines that make up the cell. Each table is identified by a key, usually a timestamp.

In general events can be of two kinds, there are events that are relative to the status of the schedule, and there are other events that are relative to the status of a task itself.

In 2015, Dr. Andrea Rosà, Lydia Y. Chen and Prof. Walter Binder published a research paper titled *Understanding the Dark Side of Big Data Clusters: An Analysis beyond Failures*[2] in which they performed several analysis on unsuccessful executions in the Google's 2011 Borg cluster traces with the aim of identifying their resource waste, their impacts on the performance of the application, and any causes that may lie behind such failures. The salient conclusion of that research is that actually lots of computations performed by Google would eventually end in failure, then leading to large amounts of computational power being wasted.

Figure 2 shows the expected transitions between event types.

2.2 Traces contents

The traces provided by Google contain mainly a collection of job and task events spanning a month of execution of the 8 different clusters. In addition to this data, some additional data on the machines' configuration in terms of resources (i.e. amount of CPU and RAM) and additional machine-related metadata.

Type code	Description
EVICT	The job or task was terminated in order to free computational resources for an higher priority job
FAIL	The job or task terminated its execution unsuccesfully due to a failure
FINISH	The job or task terminated succesfully
KILL	The job or task terminated its execution because of a manual request to stop it

Figure 1. Overview of job and task event types.

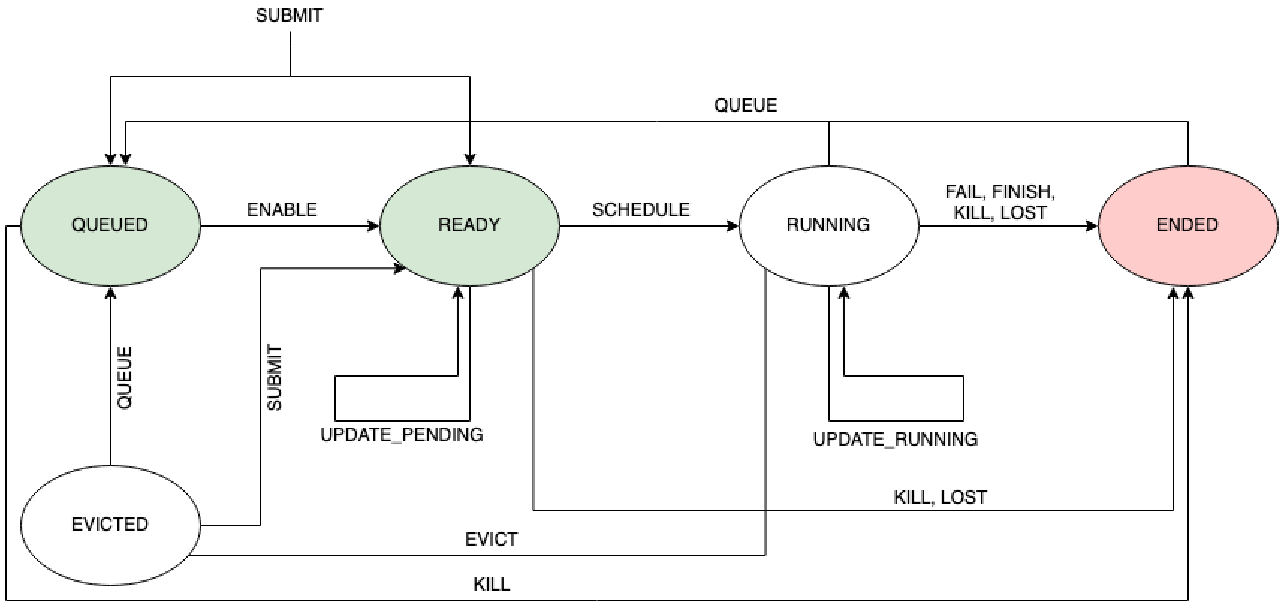


Figure 2. Typical transitions between task/job event types according to Google

Due to Google's policy, most identification related data (like job/task IDs, raw resource amounts and other text values) were obfuscated prior to the release of the traces. One obfuscation that is noteworthy in the scope of this thesis is related to CPU and RAM amounts, which are expressed respetively in NCUs (*Normalized Compute Units*) and NMUs (*Normalized Memory Units*).

NCUs and NMUs are defined based on the raw machine resource distributions of the machines within the 8 clusters. A machine having 1 NCU CPU power and 1 NMU memory size has the maximum amount of raw CPU power and raw RAM size found in the clusters. While RAM size is measured in bytes for normalization purposes, CPU power was measured in GCU (*Google Compute Units*), a proprietary CPU power measurement unit used by Google that combines several parameters like number of processors and cores, clock frequency, and architecture (i.e. ISA).

2.3 Overview of traces' format

The traces have a collective size of approximately 8TiB and are stored in a Gzip-compressed JSONL (JSON lines) format, which means that each table is represented by a single logical "file" (stored in several file segments) where each carriage return separated line represents a single record for that table.

There are namely 5 different table "files":

`machine_configs`, which is a table containing each physical machine's configuration and its evolution over time;

`instance_events`, which is a table of task events;

`collection_events`, which is a table of job events;

`machine_attributes`, which is a table containing (obfuscated) metadata about each physical machine and its evolution over time;

`instance_usage`, which contains resource (CPU/RAM) measures of jobs and tasks running on the single machines.

The scope of this thesis focuses on the tables `machine_configs`, `instance_events` and `collection_events`.

2.4 Remark on traces size

While the 2011 Google Borg traces were relatively small, with a total size in the order of the tens of gigabytes, the 2019 traces are quite challenging to analyze due to their sheer size. As stated before, the traces have a total size of 8 TiB when stored in the format provided by Google. Even when broken down to table "files", unitary sizes still reach the single tebibyte mark (namely for `machine_configs`, the largest table in the trace).

Due to this constraints, a careful data engineering based approach was used when reproducing the 2015 DSN paper analysis. Bleeding edge data science technologies like Apache Spark were used to achieve efficient and parallelized computations. This approach is discussed with further detail in the following section.

3 Project requirements and analysis

TBD (describe our objective with this analysis in detail) The aim of this thesis is to repeat the analysis performed in 2015 on the dataset Google has released in 2019 in order to find similarities and differences with the previous analysis, and ultimately find whether computational power is indeed wasted in this new workload as well. The 2019 data comes from 8 Borg cells spanning 8 different datacenters located in different geographical positions, all focused on computational oriented workloads. The data collection time span matches the entire month of May 2019.

4 Analysis methodology

Due to the inherent complexity in analyzing traces of this size, novel bleeding-edge data engineering tecniques were adopted to performed the required computations. We used the framework Apache Spark to perform efficient and parallel Map-Reduce computations. In this section, we discuss the technical details behind our approach.

4.1 Introduction on Apache Spark

Apache Spark is a unified analytics engine for large-scale data processing. In layman's terms, Spark is really useful to parallelize computations in a fast and streamlined way.

In the scope of this thesis, Spark was used essentially as a Map-Reduce framework for computing aggregated results on the various tables. Due to the sharded nature of table "files", Spark is able to spawn a thread per file and run computations using all processors on the server machines used to run the analysis.

Spark is also quite powerful since it provides automated thread pooling services, and it is able to efficiently store and cache intermediate computation on secondary storage without any additional effort required from the data engineer. This feature was especially useful due to the sheer size of the analyzed data, since the computations required to store up to 1TiB of intermediate data on disk.

The chosen programming language for writing analysis scripts was Python. Spark has very powerful native Python bindings in the form of the *PySpark* API, which were used to implement the various queries.

4.2 Query architecture

4.2.1 Overview

In general, each query written to execute the analysis follows a general Map-Reduce template.

Traces are first read, then parsed, and then filtered by performing selections, projections and computing new derived fields. After this preparation phase, the trace records are often passed through a `groupby()` operation, which by choosing one or many record fields sorts all the records into several "bins" containing records with matching values for the selected fields. Then, a map operation is applied to each bin in order to derive some aggregated property value for each grouping. Finally, a reduce operation is applied to either further aggregate those computed properties or to generate an aggregated data structure for storage purposes.

4.2.2 Parsing table files

As stated before, table "files" are composed of several Gzip-compressed shards of JSONL record data. The specification for the types and constraints of each record is outlined by Google in the form of a protobuffer specification file found in the trace release package[5]. This file was used as the oracle specification and was a critical reference for writing the query code that checks, parses and carefully sanitizes the various JSONL records prior to actual computations.

The JSONL encoding of traces records is often performed with non-trivial rules that required careful attention. One of these involved fields that have a logically-wise "zero" value (i.e. values like "0" or the empty string). For these values the key-value pair in the JSON object is outright omitted. When reading the traces in Apache Spark is therefore necessary to check for this possibility and insert back the omitted record attributes.

4.2.3 The queries

Most queries use only two or three fields in each trace records, while the original table records often are made of a couple of dozen fields. In order to save memory during the query, a projection is often applied to the data by the means of a `.map()` operation over the entire trace set, performed using Spark's RDD API.

Another operation that is often necessary to perform prior to the Map-Reduce core of each query is a record filtering process, which is often motivated by the presence of incomplete data (i.e. records which contain fields whose values is unknown). This filtering is performed using the `.filter()` operation of Spark's RDD API.

The core of each query is often a `groupby()` followed by a `map()` operation on the aggregated data. The `groupby()` groups the set of all records into several subsets of records each having something in common. Then, each of this small clusters is reduced with a `map()` operation to a single record. The motivation behind this way of computing data is that for the analysis in this thesis it is often necessary to analyze the behaviour w.r.t. time of either task or jobs by looking at their events. These queries are therefore implemented by `groupby()`-ing records by task or job, and then `map()`-ing each set of event records sorting them by time and performing the desired computation on the obtained chronological event log.

Sometimes intermediate results are saved in Spark's parquet format in order to compute and save intermediate results beforehand.

4.3 Query script design

In this section we aim to show the general complexity behind the implementations of query scripts by explaining in detail some sampled scripts to better appreciate their behaviour.

4.3.1 The "task slowdown" query script

One example of analysis script with average complexity and a pretty straightforward structure is the pair of scripts `task_slowdown.py` and `task_slowdown_table.py` used to compute the "task slowdown" tables (namely the tables in figure 7).

"Slowdown" is a task-wise measure of wasted execution time for tasks with a `FINISH` termination type. It is computed as the total execution time of the task divided by the execution time actually needed to complete the task (i.e. the total time of the last execution attempt, successful by definition).

The analysis requires to compute the mean task slowdown for each task priority value, and additionally compute the percentage of tasks with successful terminations per priority. The query therefore needs to compute the execution time of each execution attempt for each task, determine if each task has successful termination or not, and finally combine this data to compute slowdown, mean slowdown and ultimately the final table found in figure 7.

Figure 3 shows a schematic representation of the query structure.

The query first starts reading the `instance_events` table, which contains (among other data) all task event logs containing properties, event types and timestamps. As already explained in the previous section, the logical table file is actually stored as several Gzip-compressed JSONL shards. This is very useful for processing purposes, since Spark is able to parse and load in memory each shard in parallel, i.e. using all processing cores on the server used to run the queries.

After loading the data, a selection and a projection operation are performed in the preparation phase so as to "clean up" the records and fields that are not needed, leaving only useful information to feed in the "group by" phase. In this query, the selection phase removes all records that do not represent task events or that contain an unknown task ID or a null event timestamp. In the 2019 traces it is quite common to find incomplete records, since the log process is unable to capture the sheer amount of events generated by all jobs in a exact and deterministic fashion.

Then, after the preparation stage is complete, the task event records are grouped in several bins, one per task ID. Performing this operation the collection of unsorted task event types is rearranged to form groups of task events all relating to a single task.

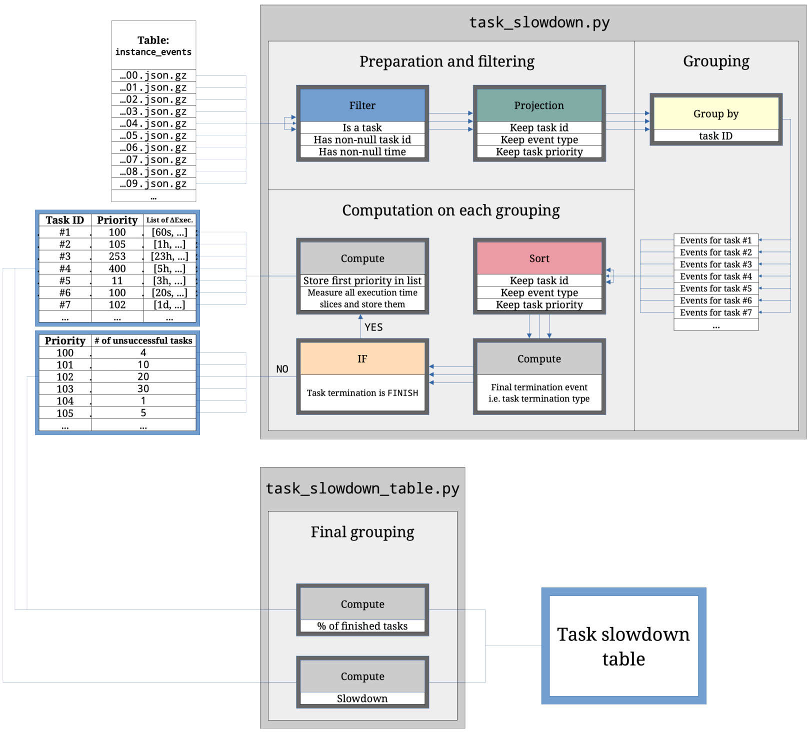


Figure 3. Diagram of the script used for the "task slowdown" query.

These obtained collections of task events are then sorted by timestamp and processed to compute intermediate data relating to execution attempt times and task termination counts. After the task events are sorted, the script iterates over the events in chronological order, storing each execution attempt time and registering all execution termination types by checking the event type field. The task termination is then equal to the last execution termination type, following the definition originally given in the 2015 Rosá et al. DSN paper.

If the task termination is determined to be unsuccessful, the tally counter of task terminations for the matching task property is increased. Otherwise, all the task termination attempt time deltas are returned. Tallies and time deltas are saved in an intermediate time file for fine-grained processing.

Finally, the `task_slowdown_table.py` processes this intermediate results to compute the percentage of successful tasks per execution and computing slowdown values given the previously computed execution attempt time deltas. Finally, the mean of the computed slowdown values is computed resulting in the clear and coincise tables found in figure 7.

4.4 Ad-Hoc presentation of some analysis scripts

TBD (with diagrams)

5 Analysis and observations

5.1 Overview of machine configurations in each cluster

Refer to figure 4.

Observations:

- machine configurations are definitely more varied than the ones in the 2011 traces
- some clusters have more machine variability

5.2 Analysis of execution time per each execution phase

Refer to figures **??** and 5.

CPU (NCU)	RAM (NMU)	Machine count	% Machines
Unknown	Unknown	8729	1.639218%
1.000000	0.500000	124234	23.329891%
0.591797	0.333496	103013	19.344801%
0.259277	0.166748	78078	14.662260%
0.708984	0.333496	55801	10.478864%
0.386719	0.333496	36237	6.804943%
0.958984	0.500000	31151	5.849843%
0.708984	0.666992	29594	5.557454%
0.386719	0.166748	27011	5.072393%
1.000000	1.000000	12286	2.307187%
0.591797	0.166748	9902	1.859496%
1.000000	0.250000	7550	1.417814%
0.958984	1.000000	3552	0.667030%
0.259277	0.333496	3024	0.567877%
0.591797	0.666992	1000	0.187790%
0.259277	0.083374	634	0.119059%
0.958984	0.250000	600	0.112674%
0.500000	0.062500	54	0.010141%
0.500000	0.250000	34	0.006385%
0.479492	0.250000	12	0.002253%
0.708984	0.250000	6	0.001127%
0.591797	0.250000	4	0.000751%
0.708984	0.500000	2	0.000376%
0.479492	0.500000	2	0.000376%

(a) All clusters

CPU (NCU)	RAM (NMU)	Machine count	% Machines
Unknown	Unknown	1377	1.623170%
0.591797	0.333496	29487	34.758469%
1.000000	0.500000	13440	15.842705%
0.708984	0.333496	12495	14.728764%
0.386719	0.333496	9057	10.676144%
0.386719	0.166748	5265	6.206238%
0.708984	0.666992	4608	5.431784%
1.000000	1.000000	4446	5.240823%
0.591797	0.166748	2484	2.928071%
0.958984	0.500000	1143	1.347337%
0.958984	1.000000	654	0.770917%
1.000000	0.250000	366	0.431431%
0.479492	0.250000	6	0.007073%
0.708984	0.250000	6	0.007073%

(b) A cluster

CPU (NCU)	RAM (NMU)	Machine count	% Machines
Unknown	Unknown	134	0.264812%
0.591797	0.333496	16184	31.982926%
1.000000	0.500000	9790	19.347061%
0.708984	0.333496	8448	16.694992%
0.958984	0.500000	5502	10.873088%
0.708984	0.666992	3832	7.572823%
1.000000	1.000000	2214	4.375321%
0.591797	0.166748	2152	4.252796%
0.386719	0.333496	816	1.612584%
0.958984	1.000000	618	1.221296%
0.591797	0.666992	500	0.988103%
0.386719	0.166748	412	0.814197%

(c) Cluster B

CPU (NCU)	RAM (NMU)	Machine count	% Machines
Unknown	Unknown	1466	2.274208%
0.259277	0.166748	15754	24.439204%
0.386719	0.333496	11104	17.225652%
0.591797	0.333496	10404	16.139741%
0.958984	0.500000	6634	10.291334%
1.000000	0.500000	5654	8.771059%
0.386719	0.166748	3580	5.553660%
0.708984	0.666992	2900	4.498774%
1.000000	1.000000	2736	4.244361%
1.000000	0.250000	2132	3.307375%
0.958984	1.000000	766	1.188297%
0.708984	0.333496	620	0.961807%
0.958984	0.250000	600	0.930781%
0.591797	0.166748	112	0.173746%

(d) Cluster C

CPU (NCU)	RAM (NMU)	Machine count	% Machines
Unknown	Unknown	498	0.794309%
0.591797	0.333496	28394	45.288376%
0.386719	0.333496	8402	13.401174%
0.259277	0.166748	8020	12.791885%
0.386719	0.166748	5806	9.260559%
0.708984	0.666992	4380	6.986092%
0.708984	0.333496	3924	6.258772%
0.591797	0.166748	2548	4.064055%
0.259277	0.333496	426	0.679469%
1.000000	0.500000	292	0.465739%
0.591797	0.250000	4	0.006380%
0.708984	0.500000	2	0.003190%

(e) Cluster D

CPU (NCU)	RAM (NMU)	Machine count	% Machines
Unknown	Unknown	536	0.671915%
0.259277	0.166748	38452	48.202377%
0.708984	0.333496	11786	14.774608%
0.958984	0.500000	8646	10.838389%
0.708984	0.666992	7606	9.534674%
1.000000	0.500000	5586	7.002457%
0.386719	0.166748	4470	5.603470%
0.259277	0.333496	1268	1.589530%
0.259277	0.083374	634	0.794765%
0.591797	0.333496	324	0.406158%
1.000000	0.250000	268	0.335957%
1.000000	1.000000	138	0.172993%
0.500000	0.062500	54	0.067693%
0.500000	0.250000	4	0.005014%

(f) Cluster E

CPU (NCU)	RAM (NMU)	Machine count	% Machines
Unknown	Unknown	1432	2.299958%
1.000000	0.500000	41340	66.396839%
0.708984	0.333496	6878	11.046866%
0.591797	0.333496	5564	8.936430%
0.958984	0.500000	2172	3.488484%
0.386719	0.166748	1544	2.479843%
0.708984	0.666992	1244	1.998008%
1.000000	0.250000	792	1.272044%
0.958984	1.000000	536	0.860878%
0.386719	0.333496	398	0.639234%
1.000000	1.000000	344	0.552504%
0.500000	0.250000	18	0.028910%

(g) Cluster F

CPU (NCU)	RAM (NMU)	Machine count	% Machines
Unknown	Unknown	1566	2.261568%
0.259277	0.166748	15852	22.892958%
1.000000	0.500000	11808	17.052741%
0.708984	0.333496	7968	11.507134%
0.591797	0.333496	7830	11.307839%
0.386719	0.166748	4690	6.773150%
0.708984	0.666992	4258	6.149269%
0.958984	0.500000	4196	6.059731%
0.386719	0.333496	3864	5.580267%
0.591797	0.166748	2606	3.763503%
1.000000	0.250000	2100	3.032754%
0.259277	0.333496	1330	1.920744%
0.958984	1.000000	778	1.123563%
1.000000	1.000000	378	0.545896%
0.500000	0.250000	12	0.017330%
0.479492	0.250000	6	0.008665%
0.479492	0.500000	2	0.002888%

(h) Cluster G

CPU (NCU)	RAM (NMU)	Machine count	% Machines
Unknown	Unknown	1720	2.933251%
1.000000	0.500000	36324	61.946178%
0.591797	0.333496	4826	8.230158%
0.708984	0.333496	3682	6.279205%
0.958984	0.500000	2858	4.873973%
0.386719	0.333496	2596	4.427163%
1.000000	1.000000	2030	3.461919%
1.000000	0.250000	1892	3.226577%
0.386719	0.166748	1244	2.121491%
0.708984	0.666992	766	1.306320%
0.591797	0.666992	500	0.852689%
0.958984	1.000000	200	0.341076%

(i) Cluster H

Figure 4. Overview of machine configurations in terms of CPU and RAM resources for each cluster

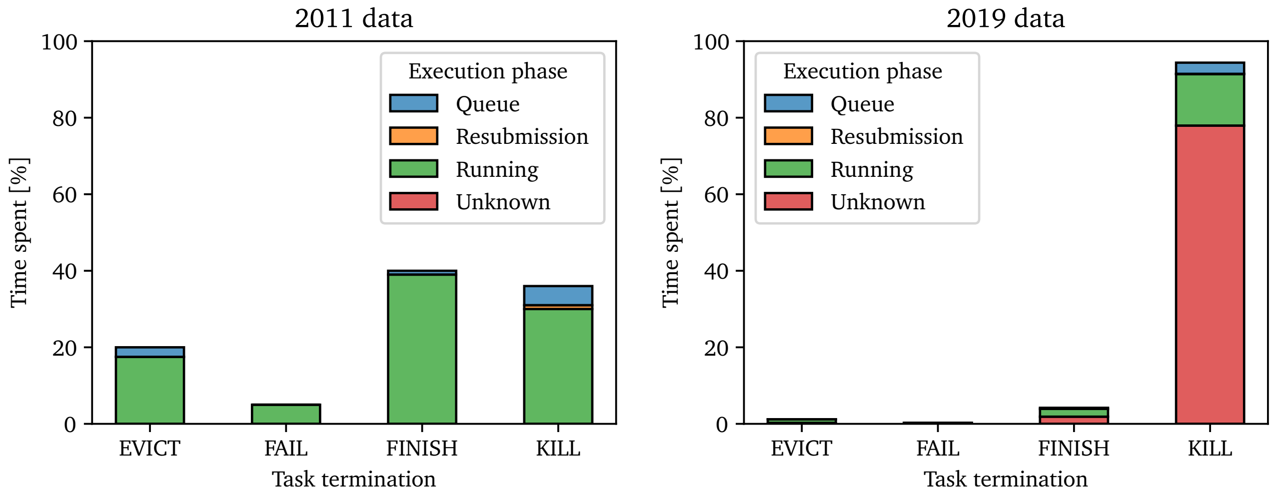


Figure 5. Relative task time (in milliseconds) spent in each execution phase w.r.t. task termination in 2011 and 2019 traces. X axis shows task termination type, Y axis shows total time % spent. Colors break down the time in execution phases. “Unknown” execution times are 2019 specific and correspond to event time transitions that are not consider “typical” by Google.

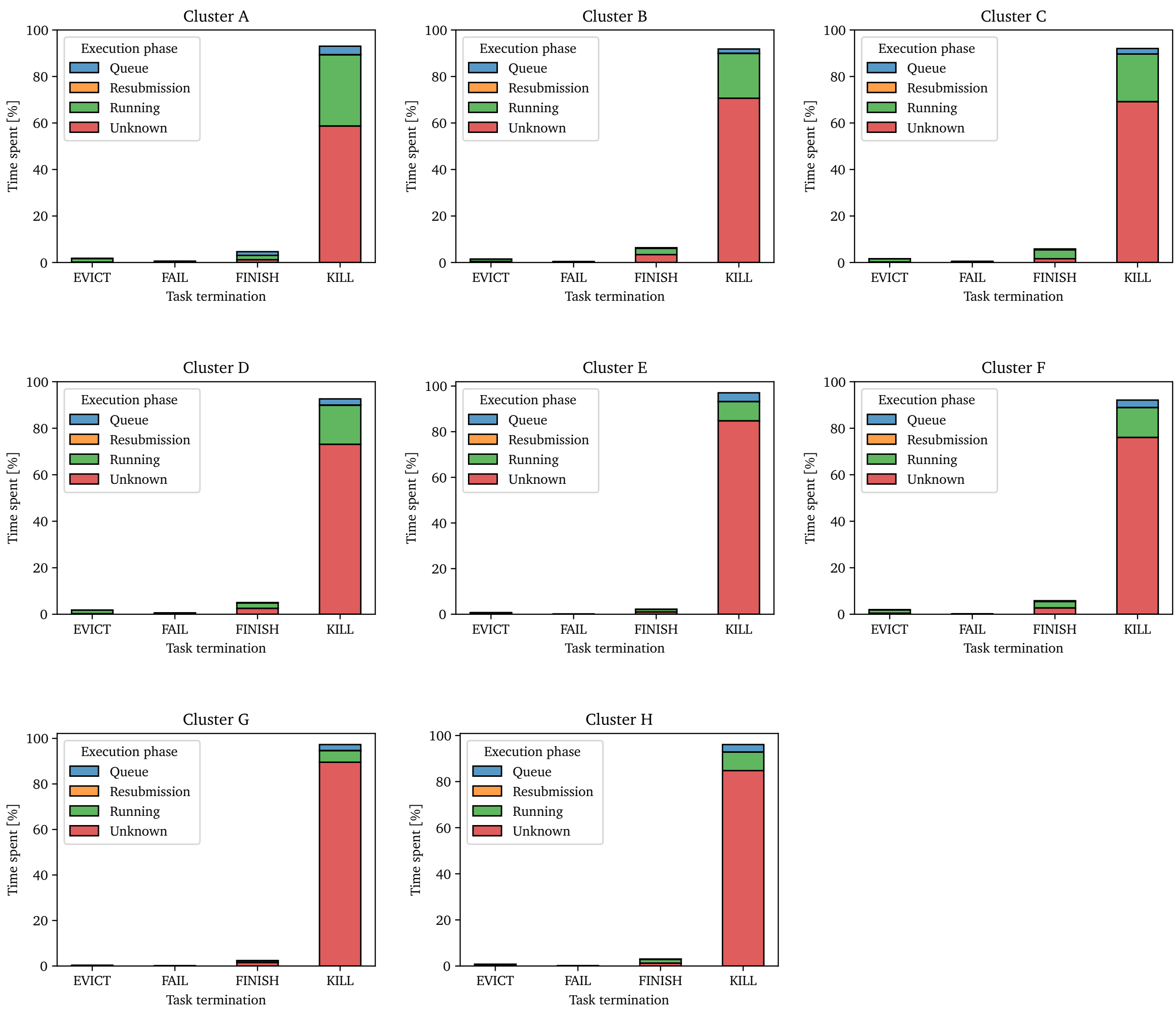


Figure 6. Relative task time (in milliseconds) spent in each execution phase w.r.t. clusters in the 2019 trace. Refer to figure 5 for axes description.

Observations:

- Across all cluster almost 50% of time is spent in “unknown” transitions, i.e. there are some time slices that are related to a state transition that Google says are not “typical” transitions. This is mostly due to the trace log being intermittent when recording all state transitions.
- 80% of the time spent in KILL and LOST is unknown. This is predictable, since both states indicate that the job execution is not stable (in particular LOST is used when the state logging itself is unstable)
- From the absolute graph we see that the time “wasted” on non-finish terminated jobs is very significant
- Execution is the most significant task phase, followed by queuing time and scheduling time (“ready” state)
- In the absolute graph we see that a significant amount of time is spent to re-schedule evicted jobs (“evicted” state)
- Cluster A has unusually high queuing times

5.3 Task slowdown

Refer to figure 7

Observations:

- Priority values are different from 0-11 values in the 2011 traces. A conversion table is provided by Google;
- For some priorities (e.g. 101 for cluster D) the relative number of finishing task is very low and the mean slowdown is very high (315). This behaviour differs from the relatively homogeneous values from the 2011 traces.
- Some slowdown values cannot be computed since either some tasks have a 0ns execution time or for some priorities no tasks in the traces terminate successfully. More raw data on those exception is in Jupyter.
- The % of finishing jobs is relatively low comparing with the 2011 traces.

5.4 Reserved and actual resource usage of tasks

Refer to figures 11 and 9.

Observations:

- Most (mesasured and requested) resources are used by killed job, even more than in the 2011 traces.
- Behaviour is rather homogeneous across datacenters, with the exception of cluster G where a lot of LOST-terminated tasks acquired 70% of both CPU and RAM

5.5 Correlation between task events’ metadata and task termination

Refer to figures 13, 15, and 17.

Observations:

- No smooth curves in this figure either, unlike 2011 traces
- The behaviour of curves for 7a (priority) is almost the opposite of 2011, i.e. in-between priorities have higher kill rates while priorities at the extremum have lower kill rates. This could also be due bt the inherent distribution of job terminations;
- Event execution time curves are quite different than 2011, here it seems there is a good correlation between short task execution times and finish event rates, instead of the U shape curve in 2015 DSN
- In figure 15 cluster behaviour seems quite uniform
- Machine concurrency seems to play little role in the event termination distribution, as for all concurrency factors the kill rate is at 90%.

5.6 Correlation between task events’ resource metadata and task termination

5.7 Correlation between job events’ metadata and job termination

Refer to figures 19, 21, and 23.

2011 priority	Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
0	Free	53.80%	2845	1767	3.37
1		67.44%	3598	2939	2.58
2	Best effort batch	90.78%	1835	1782	1.15
3		95.62%	9683	8294	3.39
4		78.05%	2006	1890	1.69
5		100%	58	58	1
6		77.99%	567	567	1.02
8		45.48%	1159	1151	1.01
9	Production	23.35%	504	496	1.07

(a) 2011 data

Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
Best effort batch	11.06%	4139	113	7.84
Free	42.85%	1374	8	1.15
Mid	2.71%	18187	157	2.55
Monitoring	2.74%	834226	130	2.05
Production	13.54%	54789	24	6.68

(b) 2019 data, aggregated

Figure 7. Mean task slowdown for each cluster and each priority "tier" for 2011 and 2019 data. **% finished** is the percentage of tasks with `FINISH` termination w.r.t. priority, **Mean response [s] (last execution)** is the mean response time (queue+execution time, in seconds) for the last task execution w.r.t. priority, **Mean response [s] (all executions)** is the response time (in seconds) of all executions, **Mean slowdown** is the mean slowdown measure w.r.t. priority. Priorities with no successfully terminated jobs have been omitted.

Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
Best effort batch	212.62%	71108	14201	5.17
Free	0.33%	5769	1203	82.97
Mid	46.22%	8510	9135	1.16
Monitoring	2.82%	1200998	1054458	2.86
Production	27.21%	4546	16845	4.12

(a) Cluster A

Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
Best effort batch	71.84%	1018454	550288	8.47
Free	45.21%	12047	5588	1.18
Mid	8.82%	225147	336262	1.11
Monitoring	4.12%	2627612	2024679	1.51
Production	30.92%	182604	466329	9.71

(b) Cluster B

Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
Best effort batch	52.96%	1236666	997117	7.40
Free	73.36%	172214	5553	1.12
Mid	95.4%	579844	248553	2.04
Monitoring	5.88%	2159459	1761833	1.74
Production	3.61%	352603	357993	4.14

(c) Cluster C

Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
Best effort batch	50.56%	1154060	1135023	12.04
Free	42.82%	22831	5506	1.15
Mid	86.34%	228762	225269	2.56
Monitoring	2.21%	1588844	913816	2.16
Production	6.53%	279565	349364	5.51

(d) Cluster D

Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
Best effort batch	0.47%	280811	205838	8.06
Free	48.15%	33050	40073	1.44
Mid	0.46%	62123	83322	10.31
Monitoring	37.71%	1415296	1263746	2.82
Production	1.96%	231639	414149	8.54

(e) Cluster E

Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
Best effort batch	44.29%	1368306	1563086	6.14
Free	45.86%	187447	37069	1.09
Mid	31.36%	200116	110201	7.60
Monitoring	8.42%	2079134	1682711	2.08
Production	3.65%	297168	492372	5.94

(f) Cluster F

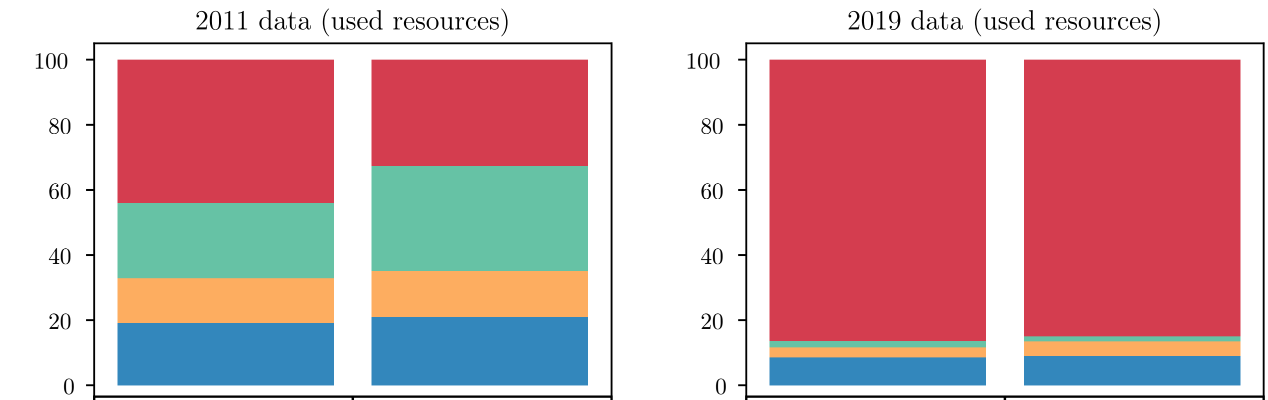
Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
Best effort batch	104.33%	294959	184724	19.06
Free	33.85%	64718	15473	1.14
Mid	49.06%	732532	706124	3.86
Monitoring	4.36%	1991341	1676276	1.72
Production	26.75%	115953	399050	14.57

(g) Cluster G

Tier	% finished	Mean response [s] (last execution)	Mean response [s] (all executions)	Mean slowdown
Best effort batch	107.03%	947368	527812	7.33
Free	28.79%	310534	290058	1.12
Mid	2.18%	338883	197440	6.49
Monitoring	4.96%	2309296	1808698	1.94
Production	2.7%	298799	470783	5.80

(h) Cluster H

Figure 8. Mean task slowdown for each cluster and each task priority for single clusters in the 2019 traces. Refer to 7 for a legend of the columns



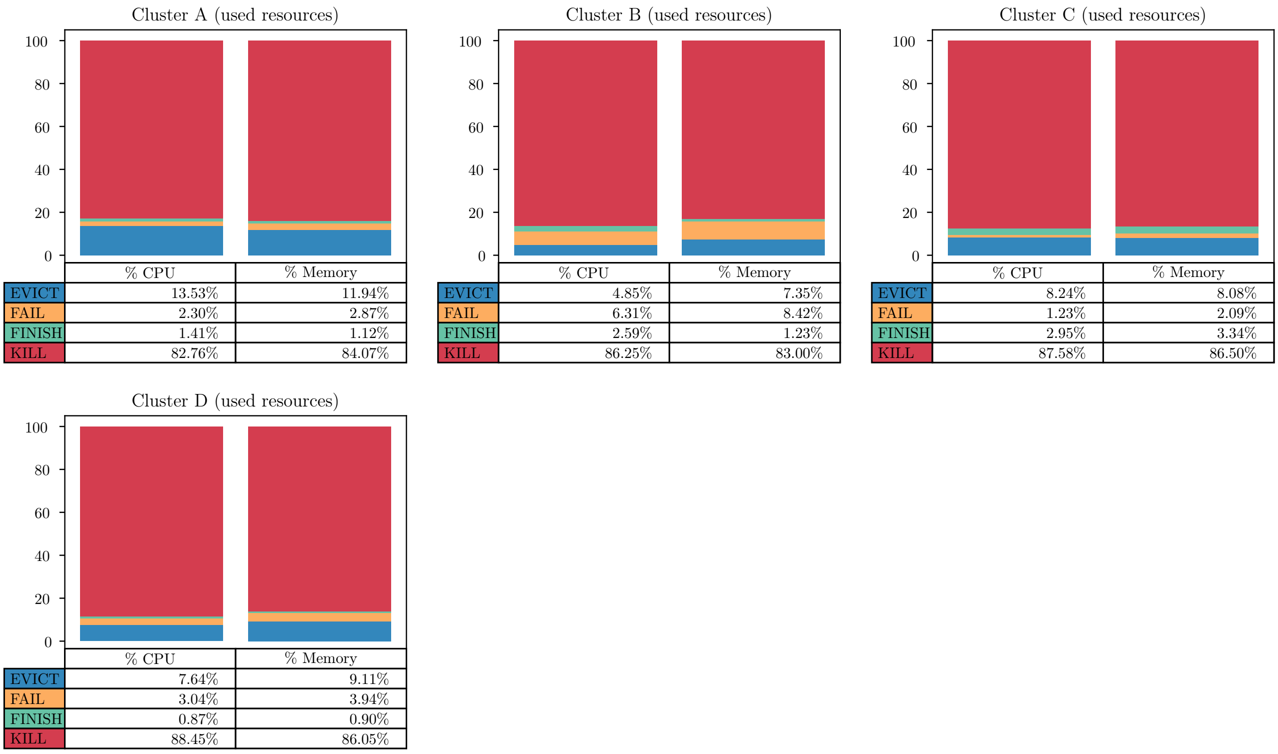
2011 data (used resources)

	% CPU	% Memory
EVICT	19.20%	21.00%
FAIL	13.70%	14.20%
FINISH	23.20%	32.10%
KILL	43.90%	32.70%

2019 data (used resources)

	% CPU	% Memory
EVICT	8.53%	9.03%
FAIL	3.17%	4.45%
FINISH	2.02%	1.66%
KILL	86.29%	84.86%

Figure 9. Percentages of CPU and RAM resources used by tasks w.r.t. task termination type in 2011 and 2019 traces (total of clusters A to D). The x axis is the type of resource, y-axis is the percentage of resource used and color represents task termination. Numeric values are displayed below the graph as a table.



Cluster A (used resources)

	% CPU	% Memory
EVICT	13.53%	11.94%
FAIL	2.30%	2.87%
FINISH	1.41%	1.12%
KILL	82.76%	84.07%

Cluster B (used resources)

	% CPU	% Memory
EVICT	4.85%	7.35%
FAIL	6.31%	8.42%
FINISH	2.59%	1.23%
KILL	86.25%	83.00%

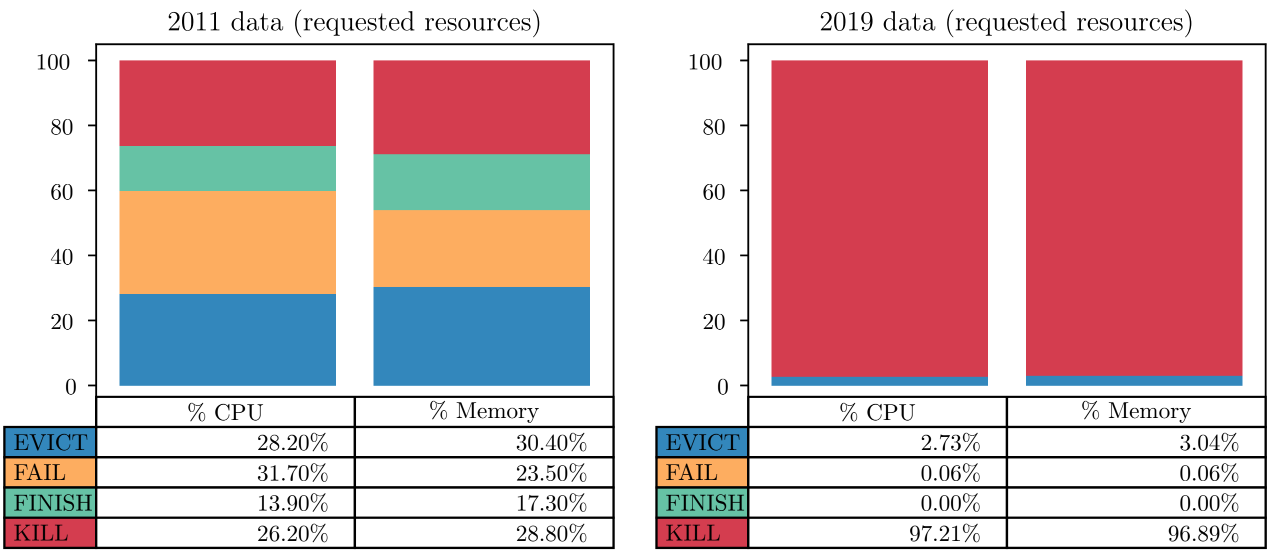
Cluster C (used resources)

	% CPU	% Memory
EVICT	8.24%	8.08%
FAIL	1.23%	2.09%
FINISH	2.95%	3.34%
KILL	87.58%	86.50%

Cluster D (used resources)

	% CPU	% Memory
EVICT	7.64%	9.11%
FAIL	3.04%	3.94%
FINISH	0.87%	0.90%
KILL	88.45%	86.05%

Figure 10. Percentages of CPU and RAM resources used by tasks w.r.t. task termination type for clusters A to D in 2019 traces. Refer to figure 9 for plot explaination.



2011 data (requested resources)

	% CPU	% Memory
EVICT	28.20%	30.40%
FAIL	31.70%	23.50%
FINISH	13.90%	17.30%
KILL	26.20%	28.80%

2019 data (requested resources)

	% CPU	% Memory
EVICT	2.73%	3.04%
FAIL	0.06%	0.06%
FINISH	0.00%	0.00%
KILL	97.21%	96.89%

Figure 11. Percentages of CPU and RAM resources requested by tasks w.r.t. task termination type in 2011 and 2019 traces. The x axis is the type of resource, y-axis is the percentage of resource used and color represents task termination. Numeric values are displayed below the graph as a table.

Cluster A (requested resources)

	% CPU	% Memory
EVICT	2.85%	3.42%
FAIL	0.06%	0.07%
FINISH	0.00%	0.00%
KILL	97.09%	96.51%

Cluster B (requested resources)

	% CPU	% Memory
EVICT	0.00%	0.01%
FAIL	0.00%	0.01%
FINISH	0.01%	0.02%
KILL	99.98%	99.97%

Cluster C (requested resources)

	% CPU	% Memory
EVICT	0.01%	0.01%
FAIL	0.00%	0.00%
FINISH	0.02%	0.02%
KILL	99.97%	99.97%

Cluster D (requested resources)

	% CPU	% Memory
EVICT	0.05%	0.01%
FAIL	0.02%	0.00%
FINISH	0.10%	0.01%
KILL	99.83%	99.98%

Cluster E (requested resources)

	% CPU	% Memory
EVICT	0.36%	0.32%
FAIL	0.05%	0.05%
FINISH	1.67%	1.31%
KILL	97.91%	98.32%

Cluster F (requested resources)

	% CPU	% Memory
EVICT	0.01%	0.01%
FAIL	0.00%	0.00%
FINISH	0.01%	0.02%
KILL	99.98%	99.96%

Cluster G (requested resources)

	% CPU	% Memory
EVICT	0.12%	0.08%
FAIL	0.02%	0.01%
FINISH	0.64%	0.54%
KILL	99.22%	99.37%

Cluster H (requested resources)

	% CPU	% Memory
EVICT	0.01%	0.00%
FAIL	0.00%	0.00%
FINISH	0.02%	0.00%
KILL	99.97%	100.00%

Figure 12. Percentages of CPU and RAM resources requested by tasks w.r.t. task termination type for in 2019 traces. Refer to figure 9 for plot explaination.

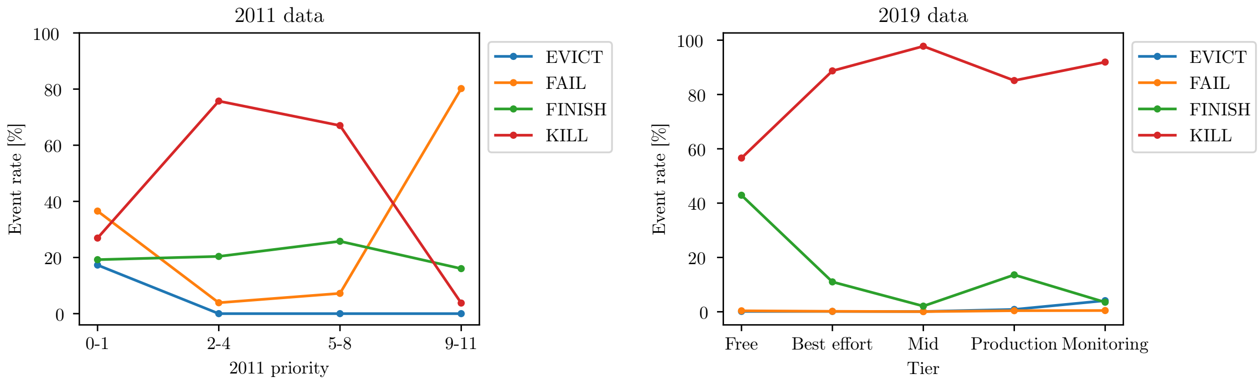


Figure 13. Task event rates vs. task priority and task termination in 2011 and 2019 (all clusters aggregated) traces. For 2019 traces tier classes instead of raw priority values are shown: 2011's $[0, 1]$ priority range corresponds to the "Free" tier, range $[2, 8]$ corresponds to the "Best effort batch" tier, range $[9-10]$ corresponds to the "Production" tier and priority 11 corresponds to the "Monitoring" tier.

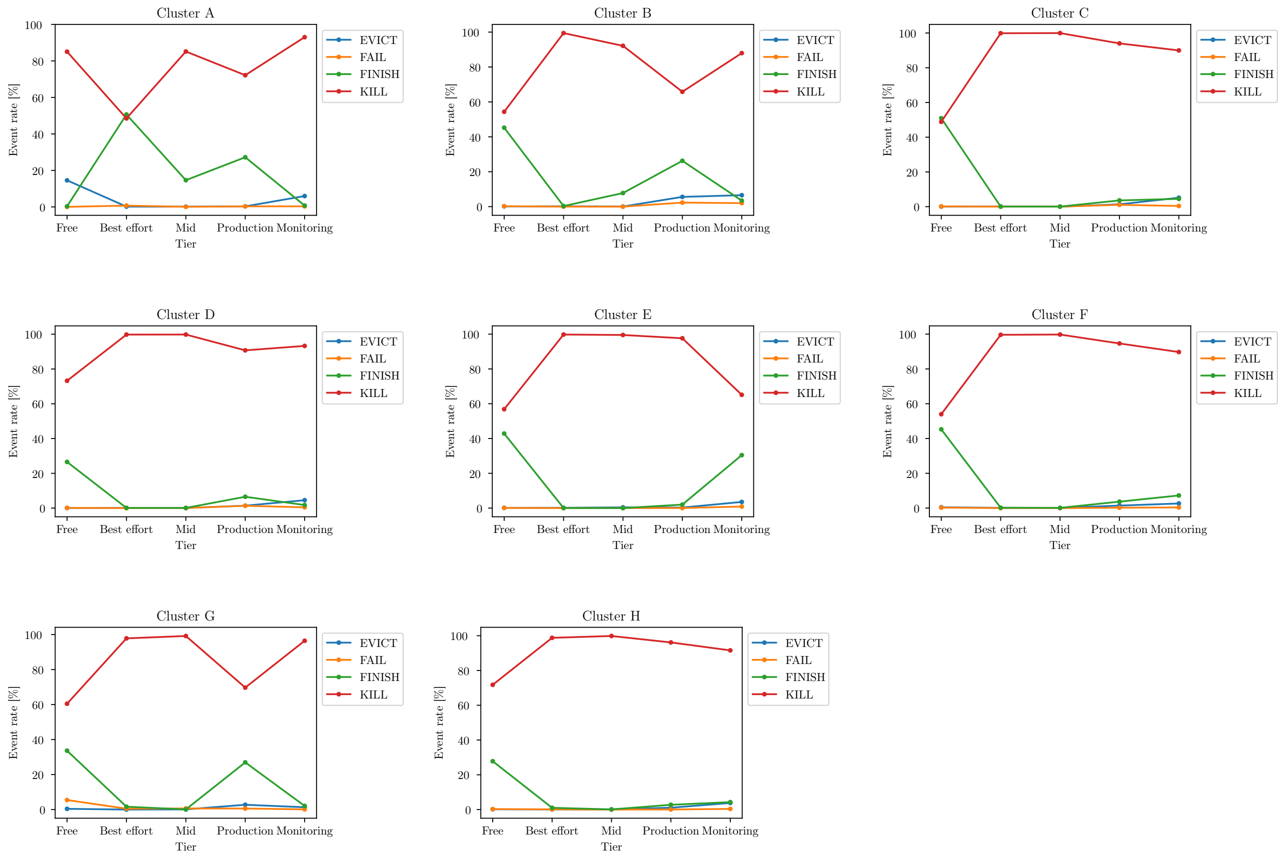


Figure 14. Task event rates vs. task priority tier and final task termination for each cluster in the 2019 traces.

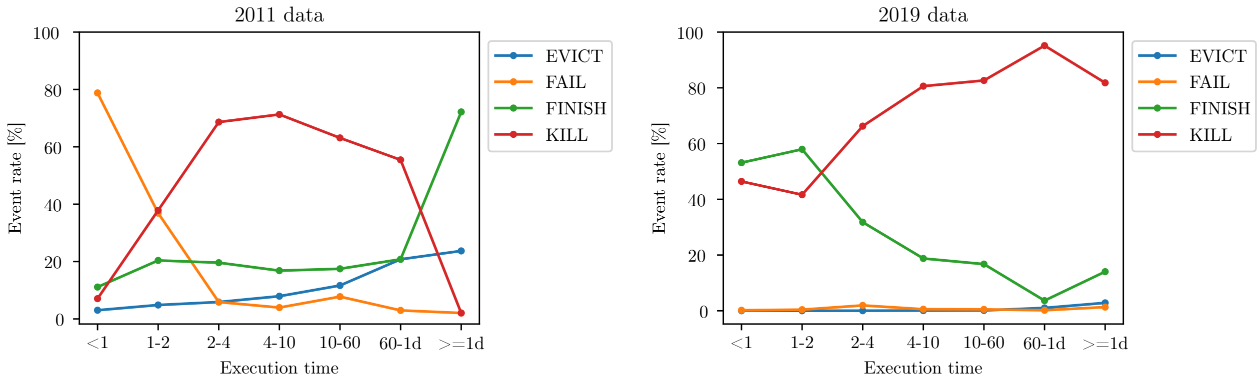


Figure 15. Task event rates vs. event execution time and final task termination in 2011 and 2019 (all clusters aggregated) traces. Execution time classes are defined in minutes, with the exception of "1d" which means one day.

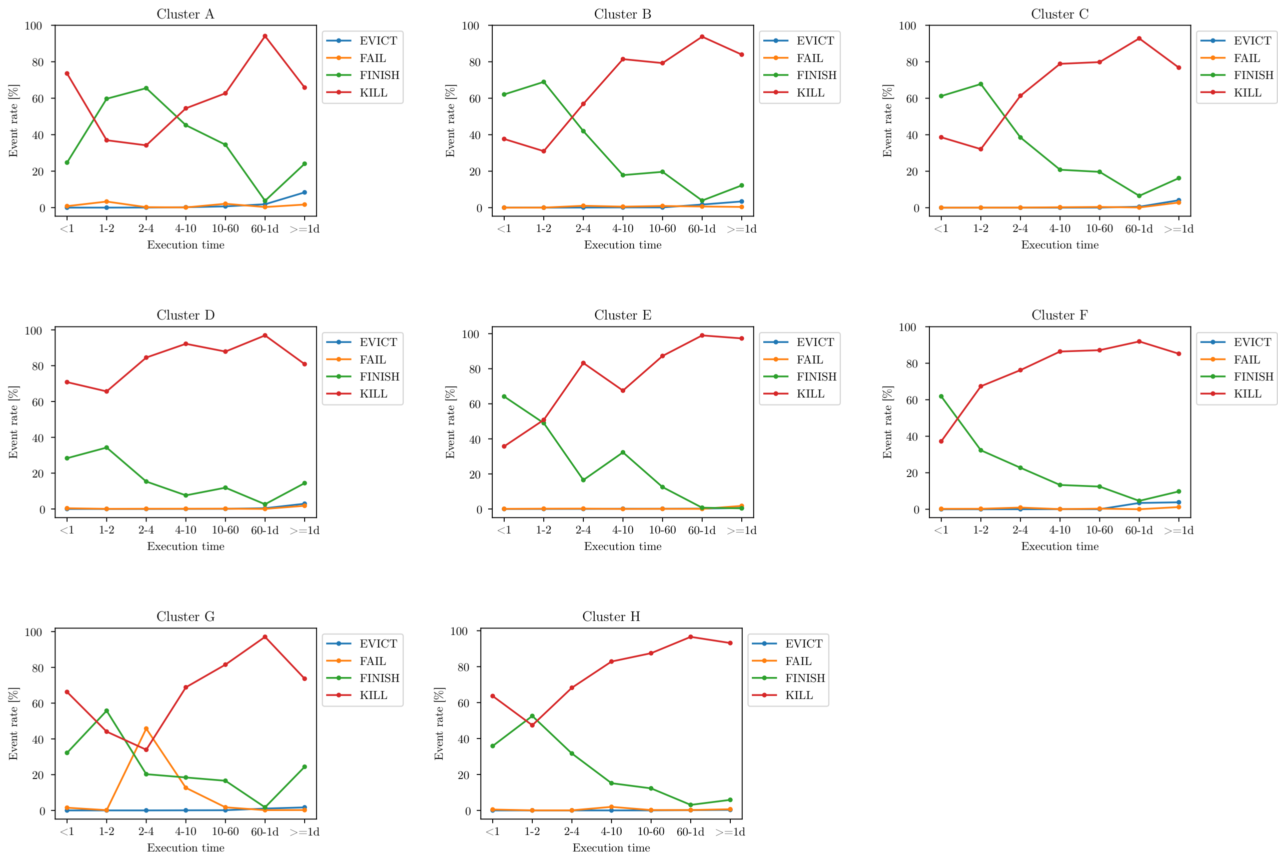


Figure 16. Task event rates vs. event execution time and final task termination for each cluster in the 2019 traces. Refer to figure 15 for interpretation of the execution time classes.

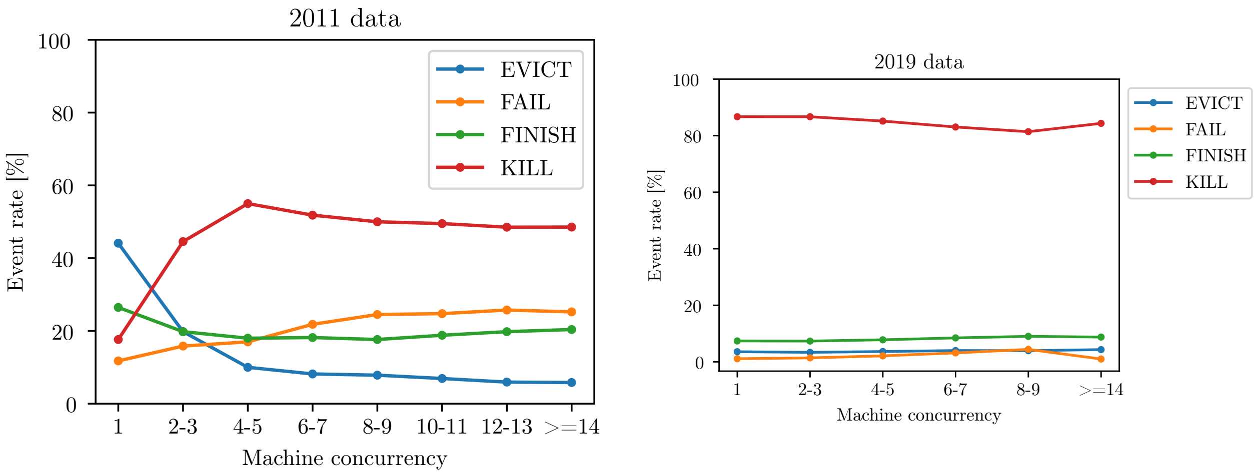


Figure 17. Task event rates vs. machine concurrency and final task termination in 2011 and 2019 (all clusters aggregated) traces. Machine concurrency is defined as the number of co-executed tasks on the machine where the analyzed task is executing.

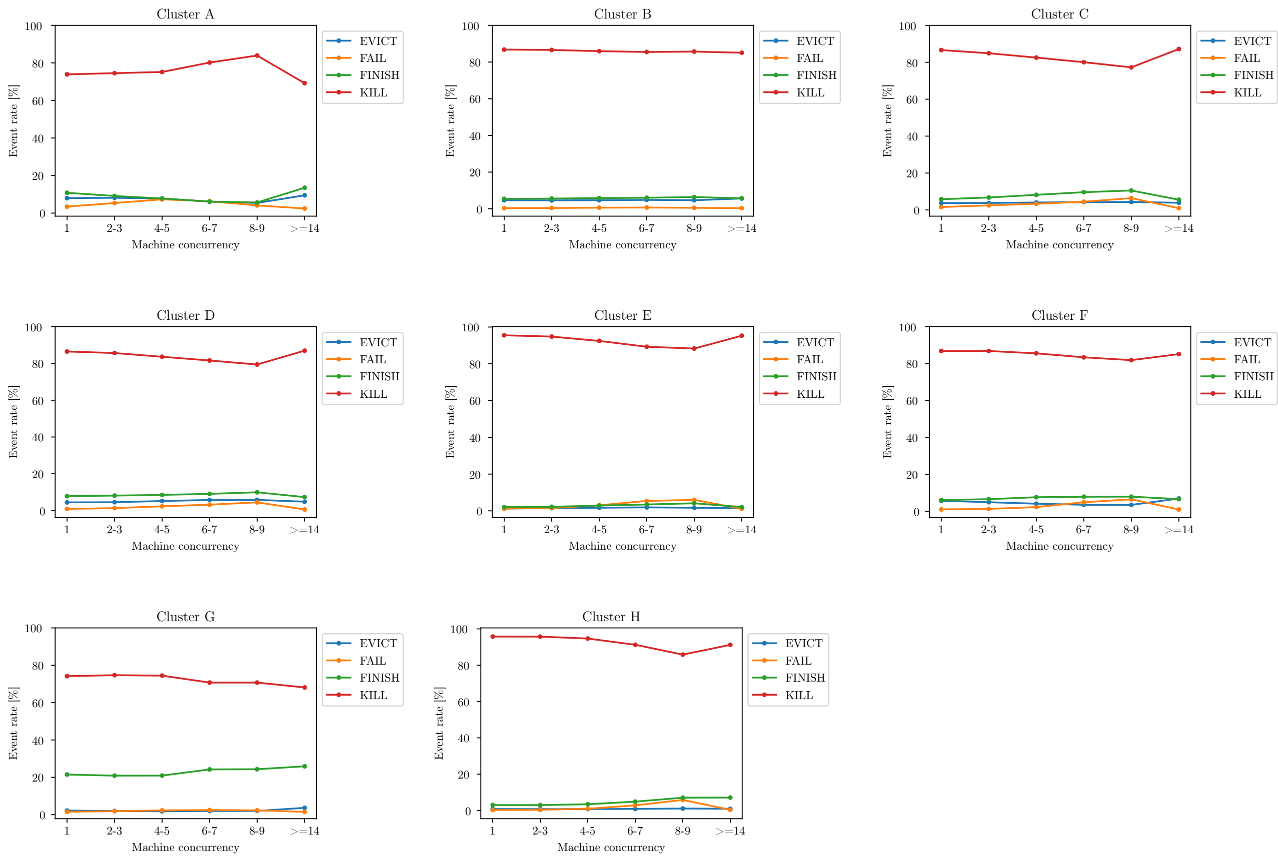


Figure 18. Task event rates vs. machine concurrency and final task termination in each 2019 cluster. Refer to figure 17 for the definition of "machine concurrency".

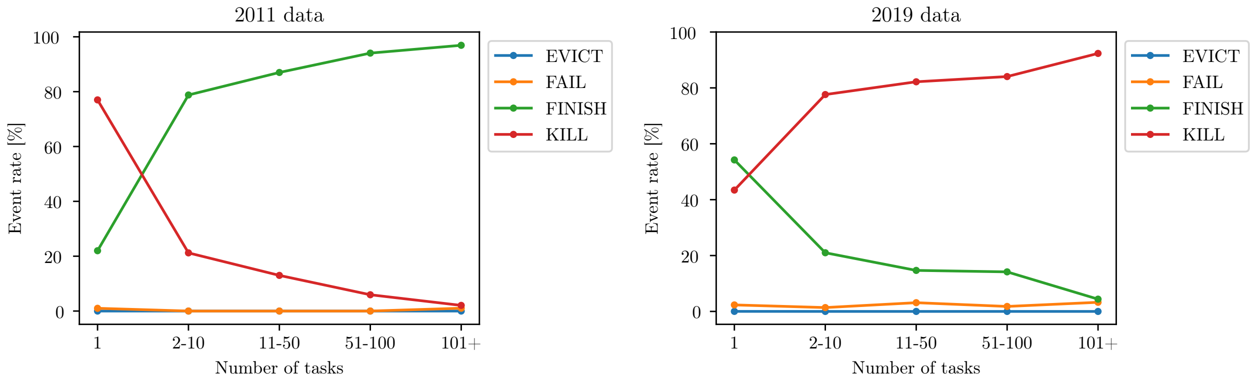


Figure 19. Job event rates vs. job size and final job termination in 2011 and 2019 (all clusters aggregated) traces. The job size is equivalent to the number of tasks belonging to the job.

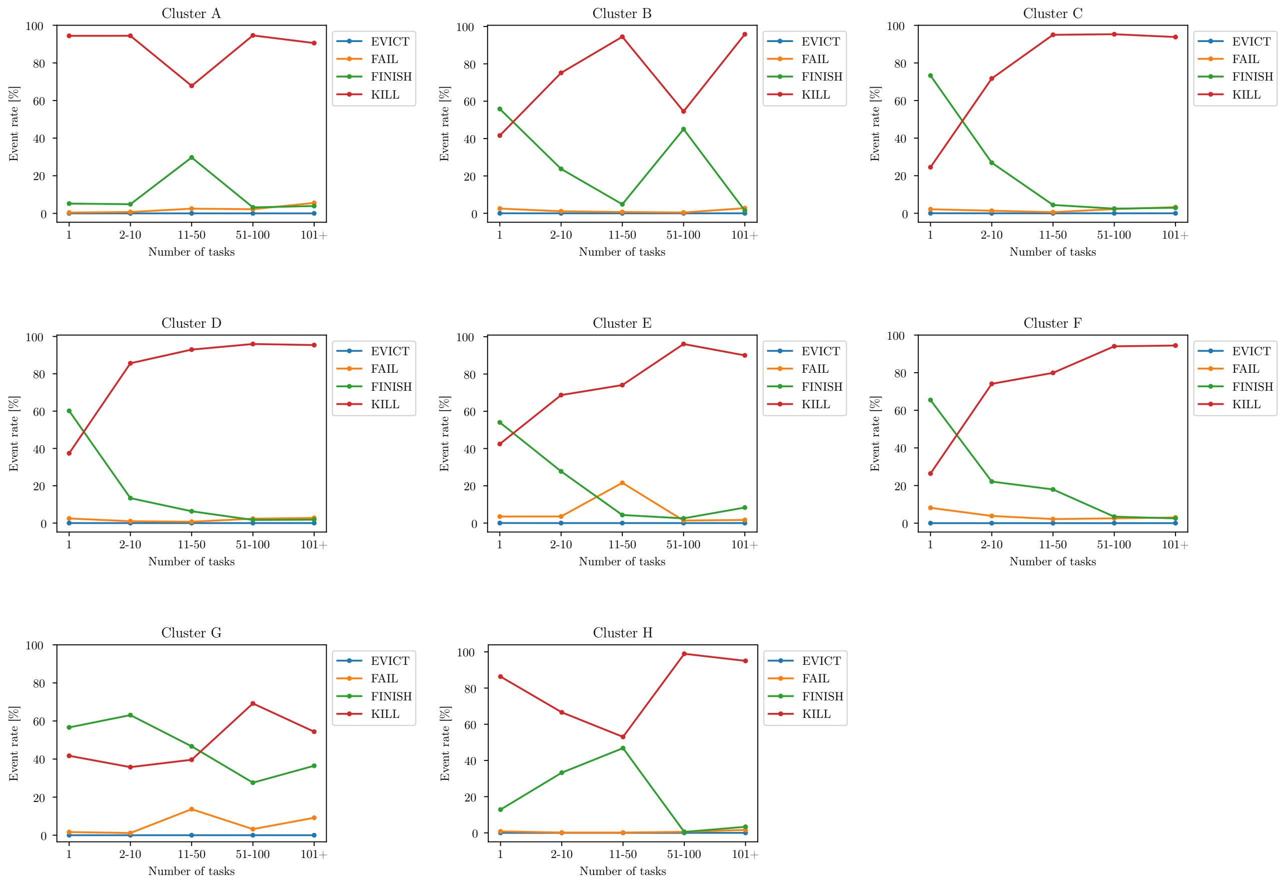


Figure 20. Job event rates vs. job size and final job termination for each 2019 cluster. Refer to figure 19 for the definition of "job size".

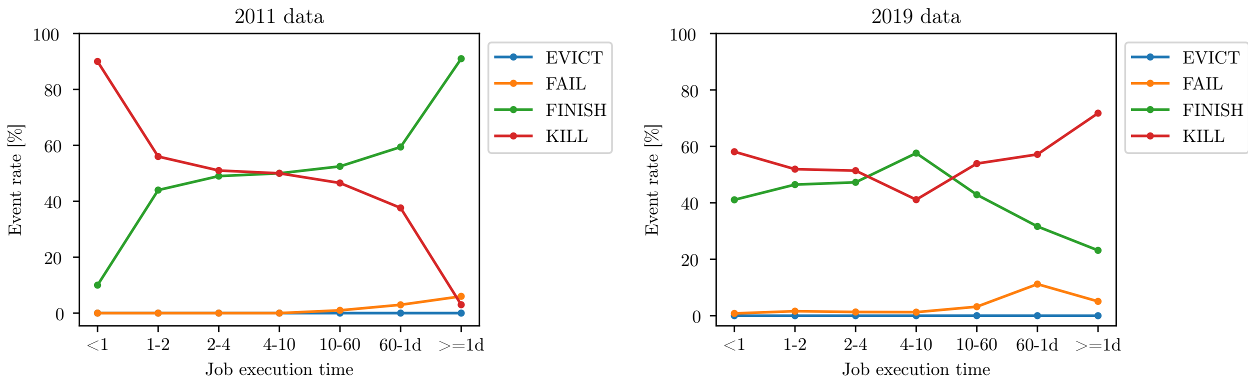


Figure 21. Job event rates vs. event execution time and final job termination in 2011 and 2019 (all clusters aggregated) traces. Execution time classes are defined in minutes, with the exception of "1d" which means one day.

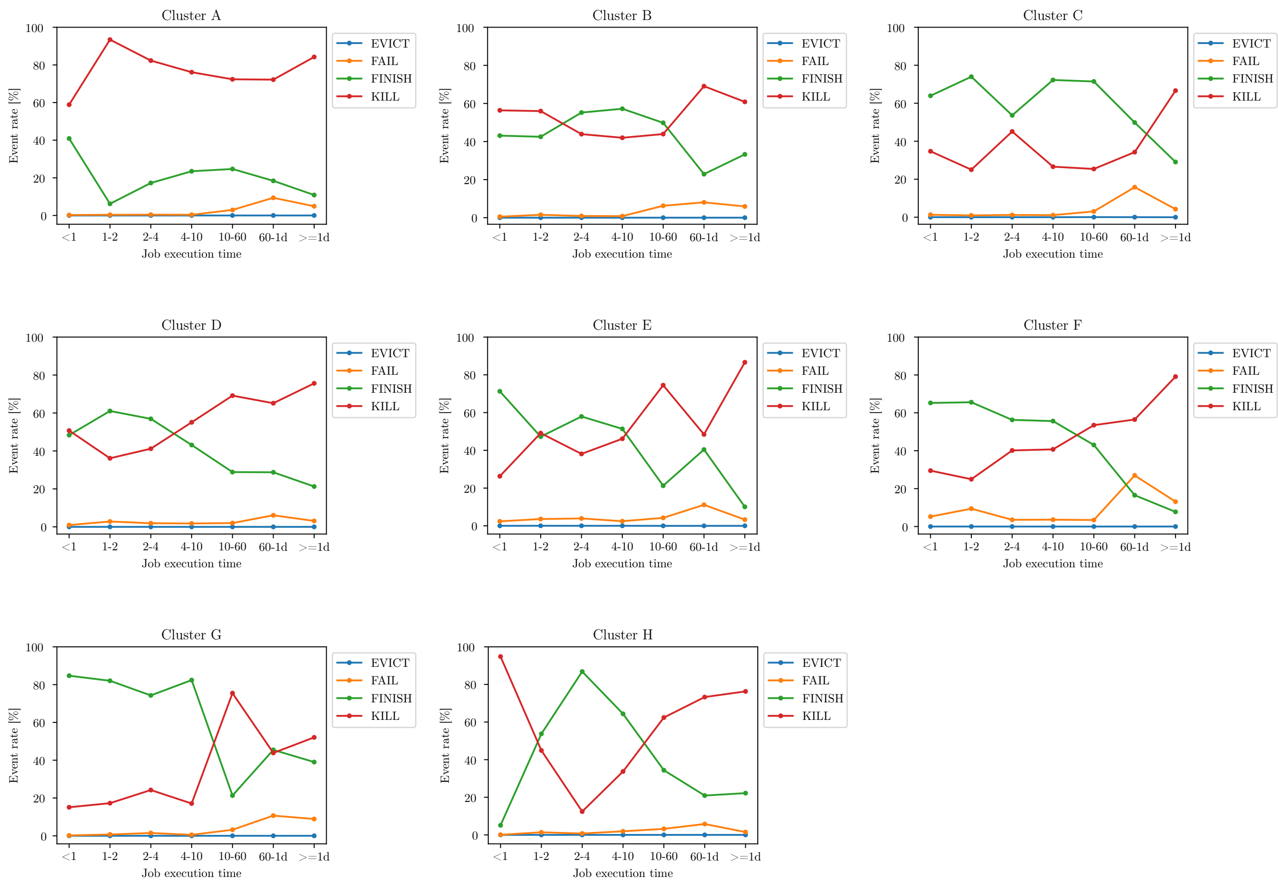


Figure 22. Job event rates vs. event execution time and final job termination for each cluster in the 2019 traces. Refer to figure 21 for interpretation of the execution time classes

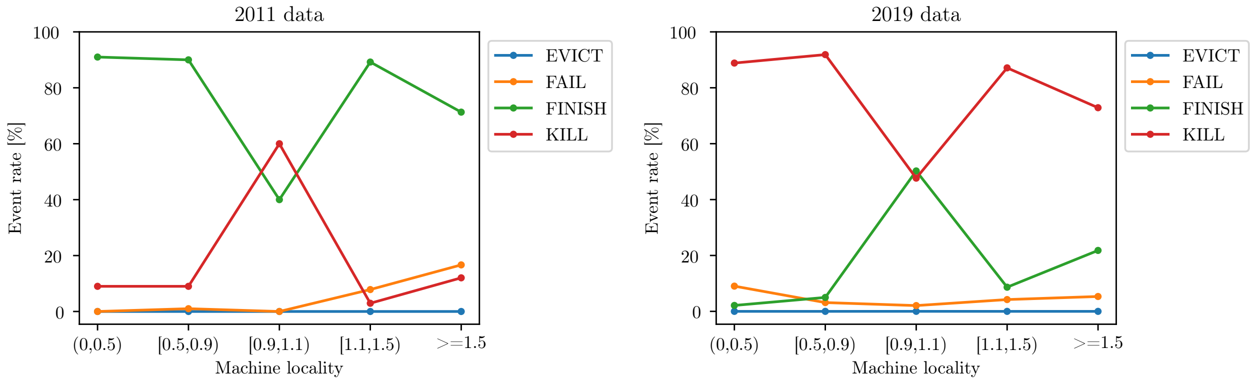


Figure 23. Job event rates vs. machine locality and final job termination in 2011 and 2019 (all clusters aggregated) traces. Machine locality is defined as the ratio between the number of distinct machines the job tasks were executed on and the job size (i.e. the number of tasks in a job).

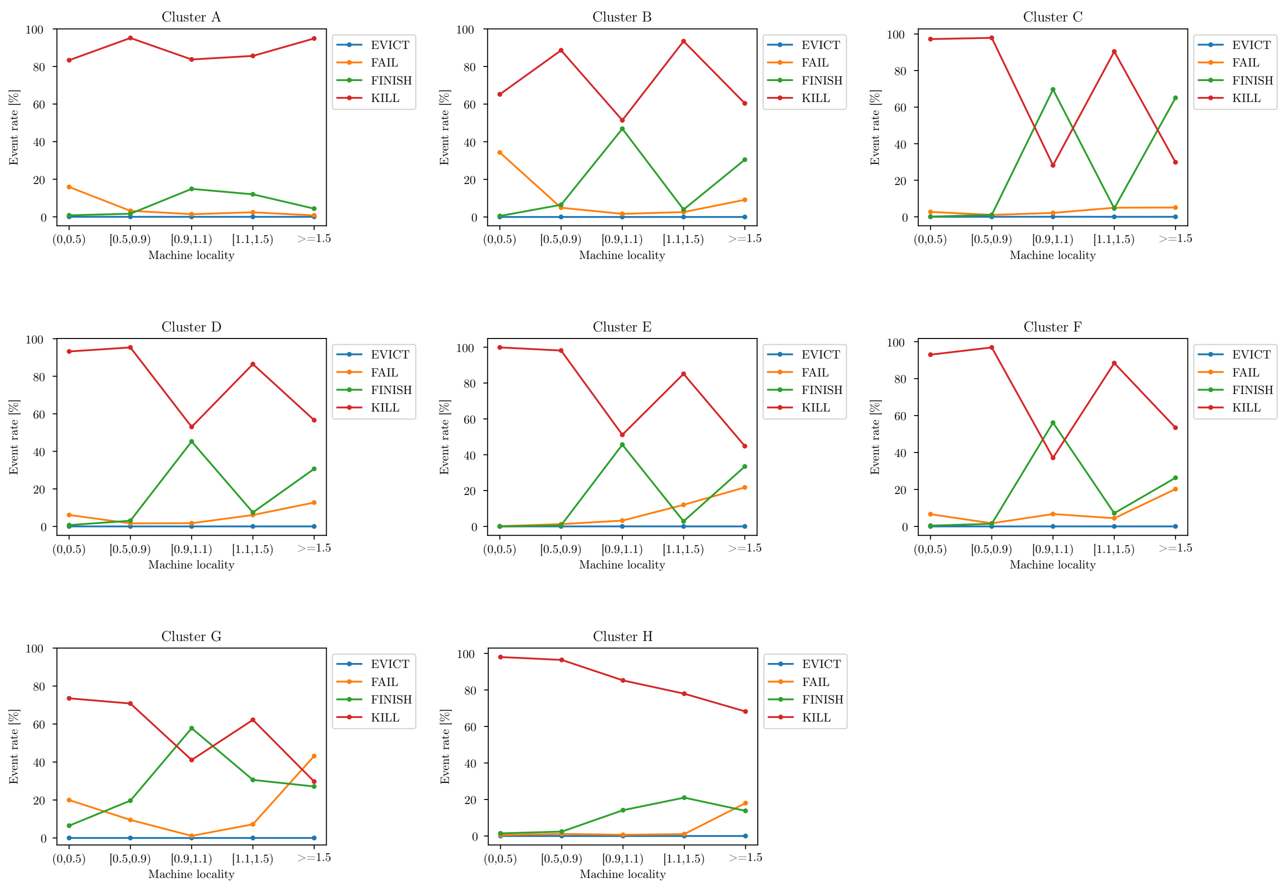


Figure 24. Job event rates vs. machine locality and final job termination for each cluster in the 2019 traces. Refer to figure 23 for the definition of "machine locality".

Observations:

- Behaviour between cluster varies a lot
- There are no “smooth” gradients in the various curves unlike in the 2011 traces
- Killed jobs have higher event rates in general, and overall dominate all event rates measures
- There still seems to be a correlation between short execution job times and successfull final termination, and likewise for kills and higher job terminations
- Across all clusters, a machine locality factor of 1 seems to lead to the highest success event rate

5.8 Mean number of tasks and event distribution per task type

Refer to figure **??**.

Observations:

- The mean number of events per task is an order of magnitude higher than in the 2011 traces
- Generally speaking, the event type with higher mean is the termination event for the task
- The # evts mean is higher than the sum of all other event type means, since it appears there are a lot more non-termination events in the 2019 traces.

5.9 Mean number of tasks and event distribution per job type

Refer to figure **??**.

Observations:

- Again the mean number of tasks is significantly higher than the 2011 traces, indicating a higher complexity of workloads
- Cluster A has no evicted jobs
- The number of events is however lower than the event means in the 2011 traces

5.10 Probability of task successful termination given its unsuccesful events

Refer to figure 29.

Observations:

- Behaviour is very different from cluster to cluster
- There is no easy conclusion, unlike in 2011, on the correlation between succesful probability and # of events of a specific type.
- Clusters B, C and D in particular have very unsmooth lines that vary a lot for small # evts differences. This may be due to an uneven distribution of # evts in the traces.

5.11 Potential causes of unsuccesful executions

TBD

6 Implementation issues – Analysis limitations

6.1 Discussion on unknown fields

TBD

6.2 Limitation on computation resources required for the analysis

TBD

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	2.372 (5)	2.094	0.259	0.004	0.015
FAIL	3.130 (8)	0.350	2.700	0.020	0.060
FINISH	2.516 (4)	0.302	1.175	1.023	0.016
KILL	1.094 (1)	0.061	0.008	0.011	1.014

(a) 2011 data

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	78.710 (342)	52.242	0.673	0.000	25.795
FAIL	24.962 (26)	0.290	23.635	0.348	0.691
FINISH	2.962 (2)	0.022	0.012	2.915	0.013
KILL	8.763 (16)	1.876	0.143	0.003	6.741

(b) 2019 data

Figure 25. Mean number of termination events and their distributions per task type between 2011 and 2019 (all clusters aggregated) traces. The tables show an overall mean accompanied by the 95-th percentile of all termination events, followed by the mean of events per event type of each termination event.

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	103.228 (719)	73.694	0.769	0.000	28.766
FAIL	11.819 (26)	0.288	11.062	0.002	0.468
FINISH	2.185 (1)	0.019	0.004	2.153	0.008
KILL	5.963 (11)	2.350	0.214	0.003	3.396

(a) Cluster A

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	83.018 (394)	64.817	0.240	0.000	17.962
FAIL	20.851 (62)	0.518	19.657	0.001	0.675
FINISH	2.995 (4)	0.020	0.021	2.943	0.012
KILL	9.173 (12)	3.351	0.276	0.004	5.541

(b) Cluster B

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	98.437 (444)	73.716	1.813	0.000	22.908
FAIL	52.010 (30)	0.773	48.446	2.035	0.756
FINISH	2.507 (2)	0.018	0.013	2.471	0.006
KILL	5.452 (6)	1.533	0.116	0.004	3.799

(c) Cluster C

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	76.759 (366)	62.001	0.700	0.000	14.058
FAIL	62.314 (62)	0.496	58.968	0.810	2.040
FINISH	3.877 (2)	0.059	0.019	3.789	0.010
KILL	6.795 (6)	1.960	0.151	0.002	4.682

(d) Cluster D

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	17.678 (72)	11.781	0.106	0.000	5.791
FAIL	112.384 (28)	0.458	111.471	0.000	0.456
FINISH	2.029 (2)	0.014	0.008	1.999	0.008
KILL	13.505 (64)	1.288	0.057	0.000	12.160

(e) Cluster E

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	70.146 (114)	23.974	0.192	0.000	45.980
FAIL	41.087 (54)	0.279	39.257	0.000	1.550
FINISH	3.129 (4)	0.019	0.004	3.008	0.098
KILL	10.288 (38)	0.384	0.098	0.001	9.804

(f) Cluster F

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	136.032 (490)	77.429	0.303	0.000	58.299
FAIL	8.948 (8)	0.016	8.593	0.000	0.339
FINISH	14.176 (2)	0.015	0.002	14.154	0.005
KILL	32.320 (164)	6.909	0.135	0.000	25.276

(g) Cluster G

Task termination	Mean number of events				
	Overall (95^{th} p.)	EVICT	FAIL	FINISH	KILL
EVICT	14.734 (40)	6.733	0.837	0.000	7.165
FAIL	41.067 (120)	0.600	37.600	0.000	2.867
FINISH	3.681 (2)	0.024	0.014	3.633	0.011
KILL	17.976 (98)	0.633	0.170	0.000	17.173

(h) Cluster H

Figure 26. Mean number of termination events and their distributions per task type for each cluster in the 2019 traces. The tables show an overall mean accompanied by the 95-th percentile of all termination events, followed by a mean of events per event type of each termination event.

Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	0.989 (1)	1.000	0.000	0.000	0.000
FAIL	43.126 (200)	0.114	2.300	0.981	12.833
FINISH	3.074 (2)	0.005	0.153	1.778	0.014
KILL	53.919 (178)	0.235	0.103	0.288	11.337

(a) 2011 data

Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	0.989 (1)	1.000	0.000	0.000	0.000
FAIL	43.126 (200)	0.114	2.300	0.981	12.833
FINISH	3.074 (2)	0.005	0.153	1.778	0.014
KILL	53.919 (178)	0.235	0.103	0.288	11.337

(b) 2019 data

Figure 27. tbd

Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	–	–	–	–	–
FAIL	90.793 (499)	0.695	0.684	0.086	1.850
FINISH	1.187 (1)	0.005	0.001	1.073	0.024
KILL	16.533 (10)	1.045	0.074	0.461	1.189

(a) Cluster A

Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0.000	0.000	0.000
FAIL	74.368 (374)	2.003	1.994	0.267	4.944
FINISH	6.304 (10)	0.022	0.008	2.349	0.013
KILL	69.853 (234)	1.696	0.158	0.614	3.009

(b) Cluster B

Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.001	0.000	0.000	0.000
FAIL	41.982 (200)	3.484	0.998	0.376	3.998
FINISH	1.991 (1)	0.022	0.017	1.565	0.017
KILL	110.681 (652)	0.627	0.059	0.656	2.267

(c) Cluster C

Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0.000	0.000	0.000
FAIL	43.356 (250)	6.112	0.949	0.531	6.498
FINISH	2.109 (2)	0.268	0.013	1.723	0.019
KILL	89.648 (283)	1.013	0.054	0.283	3.256

(d) Cluster D

Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0.000	0.000	0.000
FAIL	23.081 (25)	0.247	0.666	0.717	1.588
FINISH	7.776 (2)	0.019	0.029	1.934	0.021
KILL	88.790 (309)	0.706	0.029	0.461	7.572

(e) Cluster E

Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0.000	0.000	0.000
FAIL	17.161 (8)	0.621	0.546	0.426	7.559
FINISH	2.941 (2)	0.015	0.051	1.670	0.162
KILL	103.889 (361)	0.183	0.064	0.417	5.824

(f) Cluster F

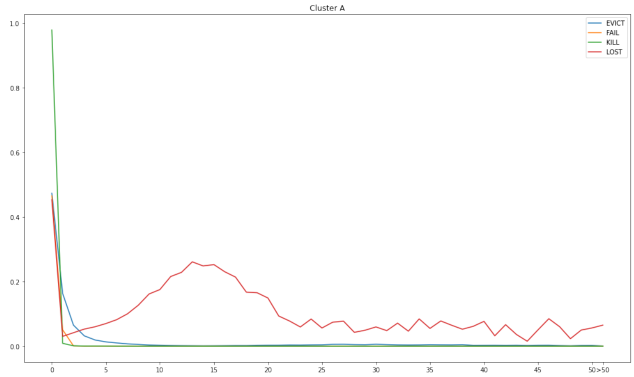
Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0.000	0.000	0.000
FAIL	51.835 (250)	0.556	3.335	0.608	20.352
FINISH	8.519 (36)	0.002	0.630	1.760	0.005
KILL	37.055 (100)	5.687	0.065	0.080	19.166

(g) Cluster G

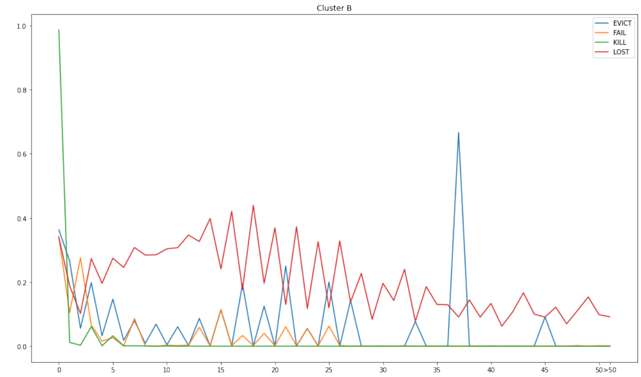
Job termination	# of tasks mean. (95^{th} p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0.000	0.000	0.000
FAIL	20.504 (1)	0.114	2.300	0.981	12.833
FINISH	4.278 (14)	0.005	0.153	1.778	0.014
KILL	11.023 (3)	0.235	0.103	0.288	11.337

(h) Cluster H

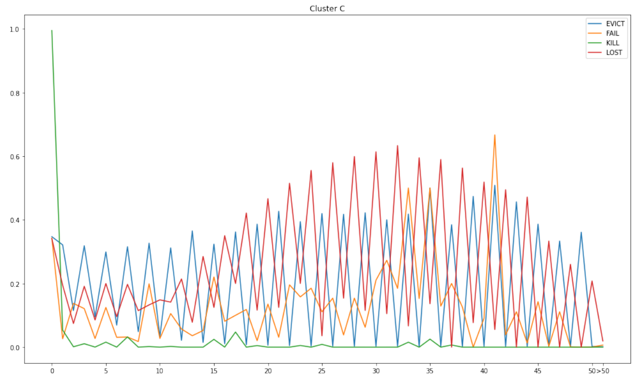
Figure 28. tbd



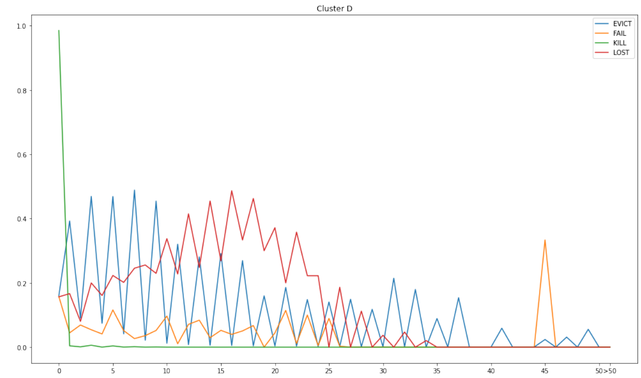
(a) Cluster A



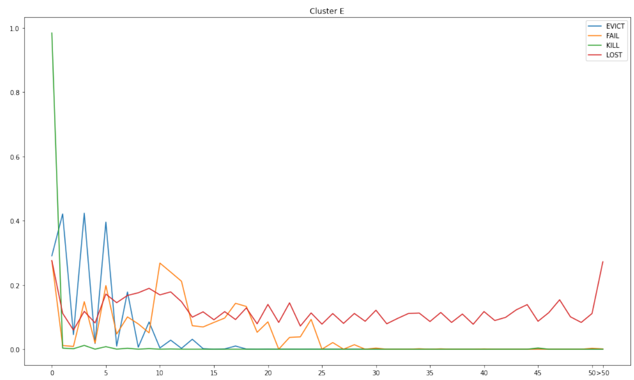
(b) Cluster B



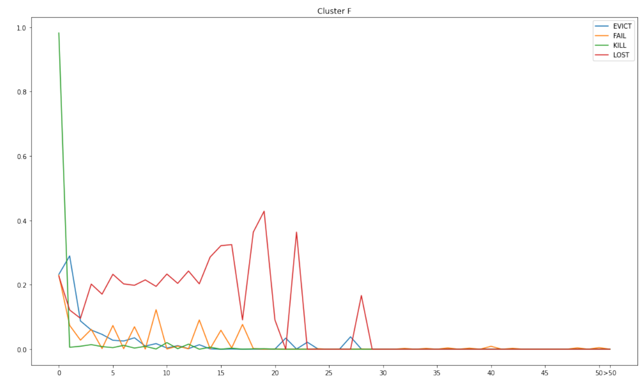
(c) Cluster C



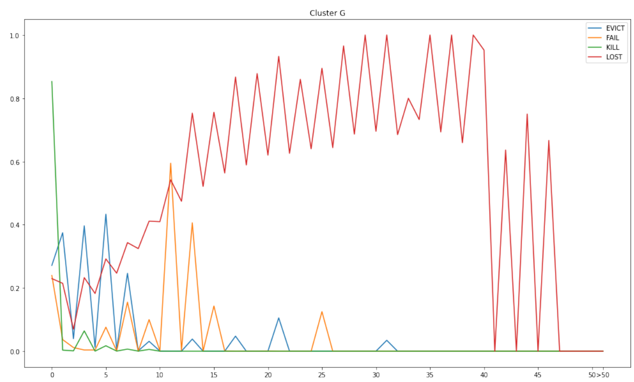
(d) Cluster D



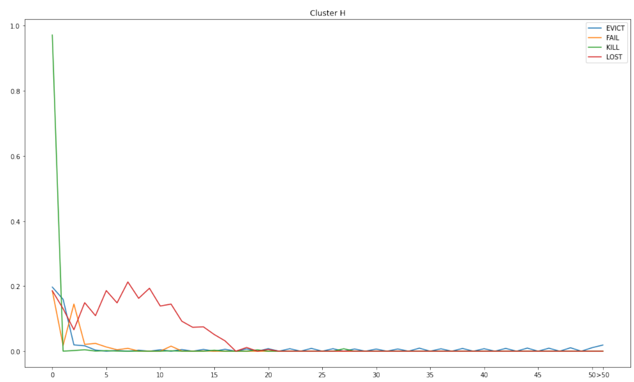
(e) Cluster E



(f) Cluster F



(g) Cluster G



(h) Cluster H

Figure 29. Conditional probability of task success given a number of specific unsuccesful events observed, i.e. eviction, fail, kill or lost.

6.3 Other limitations ...

TBD

7 Conclusions and future work or possible developments

TBD

References

[1] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, et al. "Large-scale cluster management at Google with Borg". In: *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France, 2015.

[2] Andrea Rosà, Lydia Y. Chen, and Walter Binder. "Understanding the Dark Side of Big Data Clusters: An Analysis beyond Failures". In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2015, pp. 207–218. DOI: `10.1109/DSN.2015.37`.

[3] Muhammad Tirmazi, Adam Barker, Nan Deng, et al. "Borg: the Next Generation". In: *EuroSys'20*. Heraklion, Crete, 2020.

[4] John Wilkes. *Google cluster-usage traces v3.pdf*. Aug. 2020. URL: `https://drive.google.com/file/d/10r6cnJ5cJ89fPWCgj7j4LtLBqYN9RiI9/view`.

[5] Nan Deng. *Google 2019 Borg traces protobuffer specification*. Aug. 2020. URL: `https://github.com/google/cluster-data/blob/master/clusterdata_trace_format_v3.proto`.