Università
della
Svizzera
italiana

**Faculty
of Informatics**

Bachelor Thesis

June 25, 2021

Understanding and Comparing Unsuccessful Executions in Large Datacenters

Claudio Maggioni

Abstract

The thesis 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 thesis is to compare the resource waste caused by unsuccessful executions, their impact on application performance, and their root causes. We show the strong negative impact on CPU and RAM usage and on task slowdown. We analyze patterns of unsuccessful jobs and tasks, focusing on their interdependency. Moreover, we uncover their root causes by inspecting key workload and system attributes such as machine locality and concurrency level.

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

Advisor's approval (Prof. Walter Binder, Dr. Andrea Rosà):   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" puts 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 the foundation of the 2015 Rosà 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.

Even by glancing at some of the spatial and temporal analyses performed on the Google Borg traces in this report, it is evident that unsuccessful executions play a major role in the waste of resources in clusterized computations. For examples, Figure 7 shows the distribution of machine time over "tasks" (i.e. executables running in Borg) with different termination "states", of which FINISH is the only successful one. For the 2011 Borg traces we have that more than half of the machine time is invested in carrying out non-successful executions, i.e. executing programs that would eventually "crash" and potentially not leading to useful results[1]. The 2019 subplot paints an even darker picture, with less than 5% of machine time used for successful computation.

Given that even a major player in big data computation like Google is struggling at efficiently allocating computational resources, the impact of execution failures is indeed significant and worthy of study. Given also the significance and data richness of both trace packages, the analysis performed in this report can be of interest for understanding the behaviour of failures in similar clusterized systems, and could potentially be used to build predictive models to mitigate or erase the resource impact of unsuccessful executions.

Given that the new 2019 Google Borg cluster traces are about 100 times larger than the 2011 ones, and given that the entire compressed traces package has a non-trivial size (weighing approximately 8 TiB [4]), the computations required to perform the analysis we illustrate in this report cannot be performed with classical data science techniques. A considerable amount of computational power was needed to carry out the computations, involving at their peek 3 dedicated Apache Spark servers over the span of 3 months. Additionally, the analysis scripts have been written by exploiting the power of parallel computing, following most of the time a MapReduce-like structure.

This project aims to repeat the analysis performed in 2015 DSN Rosà et al. paper [2] to highlight similarities and differences in Google Borg workload and the behaviour and patterns of executions within it. Thanks to this analysis, we aim to understand even better the causes of failures and how to prevent them. Additionally, given the technical challenge this analysis posed, the report aims to provide an overview of some basic data engineering techniques for big data applications.

The report is structured as follows. Section 2 contains information about the current state of the art for Google Borg cluster traces. Section 3 provides an overview including technical background information on the data to analyze and its storage format. Section 4 discusses about the project requirements and the data science methods used to perform the analysis. Section 5, Section 6 and Section 7 show the result obtained while analyzing, respectively the performance input of unsuccessful executions, the patterns of task and job events, and the potential causes of unsuccessful executions. Finally, Section 8 concludes.

---

[1]This is only a speculation, since both the 2011 and the 2019 traces only provide a "black box" view of the Borg cluster system. Neither of the accompanying papers for both traces [1] [3] or the documentation for the 2019 traces [4] ever mention if non-successful tasks produce any useful result.

Cluster	Timezone
A	America/New York
B	America/Chicago
C	America/New York
D	America/New York
E	Europe/Helsinki
F	America/Chicago
G	Asia/Singapore
H	Europe/Brussels

Figure 1. Approximate geographical location obtained from the datacenter's timezone of each cluster in the 2019 Google Borg traces.

2 State of the Art

In 2015, Rosà et al. published a research paper titled *Understanding the Dark Side of Big Data Clusters: An Analysis beyond Failures* [2] in which they performed several analyses 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.

However, with the release of the new 2019 traces [3], the results and conclusions found by that paper could be potentially outdated in the current large-scale computing world. The new traces not only provide updated data on Borg's workload, but provide more data as well: the new traces contain data from 8 different Borg "cells" (i.e. clusters) in datacenters across the world, from now on referred as "Cluster A" to "Cluster H".

The geographical location of each cluster can be consulted in Figure 1. The information in that table was provided by the 2019 traces documentation [4].

The new 2019 traces provide richer data even on a cluster by cluster basis. For example, the amount and variety of server configurations per cluster increased significantly from 2011. An overview of the machine configurations in the cluster analyzed with the 2011 traces and in the 8 clusters composing the 2019 traces can be found in Figure 2 and in Figure 3 on a cluster-by-cluster basis.

There are two main works covering the new data, one being the paper *Borg: The Next Generation* [3], which compares the overall features of the trace with the 2011 one [1] [5], and one covering the features and performance of *Autopilot* [6], a software that provides autoscaling features in Borg. The new traces have also been analyzed from the execution priority perspective [7], as well as from a cluster-by-cluster comparison [8] given the multi-cluster nature of the new traces.

Other studies have been performed in similar big-data systems focusing on the failure of hardware components and software bugs [9] [10] [11] [12].

However, the community has not yet performed any research on the new Borg traces analysing unsuccessful executions, their possible causes, and the relationships between tasks and jobs. Therefore, the only current research in this field is this very report, providing and update to the the 2015 Rosà et al. paper [2] focusing on the new trace.

Figure 2. Overview of machine configurations in term of CPU and Memory power in 2011 and 2019 (all clusters aggregated) traces. In the 2019 traces NCU stands for "Normalized Compute Unit" and NMU stands for "Normalized Memory Unit": both are [0, 1] normalizations of resource values. While memory was measured in terms of capacity, CPU power was measured in "Google Compute Units" (GCUs), an opaque umbrella metric used by Google that factors in CPU clock, number of cores/processors, and CPU ISA architecture.

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%

(a) 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%

(b) 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%

(c) 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%

(d) 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%

(e) 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%

(f) 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%

(g) 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%

(h) Cluster H

Figure 3. Overview of machine configurations in terms of CPU and RAM resources for each cluster in the 2019 traces. Refer to Figure 2 for a column legend.

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 4. Overview of job and task termination event types.

3 Background

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.

Both tasks and jobs lifecyles are represented by several events, which are encoded and stored in the trace as rows of various tables. Among the information events provide, the field "type" provides information on the execution status of the job or task. We focus only on events whose "types" indicate a termination, i.e. the end of a task or job's execution. These termination event types are illustrated in Figure 4. We then define an unsuccessful execution to be an execution characterized by a termination event of type EVICT, FAIL or KILL. Conversely, a successful execution is characterized by a FINISH termination event.

3.1 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.

Figure 5 shows the expected transitions between event types.

3.2 Trace 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.

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).

3.3 Trace 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
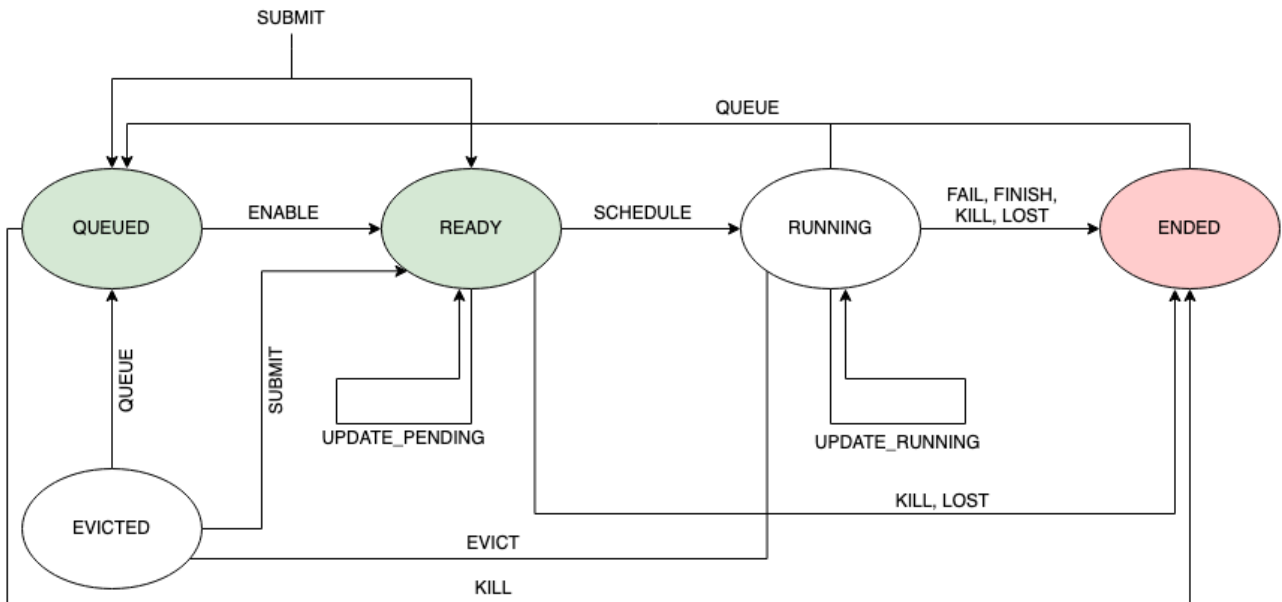
Figure 5. Typical transitions between task/job event types according to Google

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.

3.4   Remark on Trace 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.

4 Project Requirements and Analysis Methodology

The aim of this project 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.

Due to the inherent complexity in analyzing traces of this size, non-trivial 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 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 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 [13]. 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 and the *Task Slowdown* Script

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.

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 9).

"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 9.

Figure 6 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.

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 Rosà 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 9.
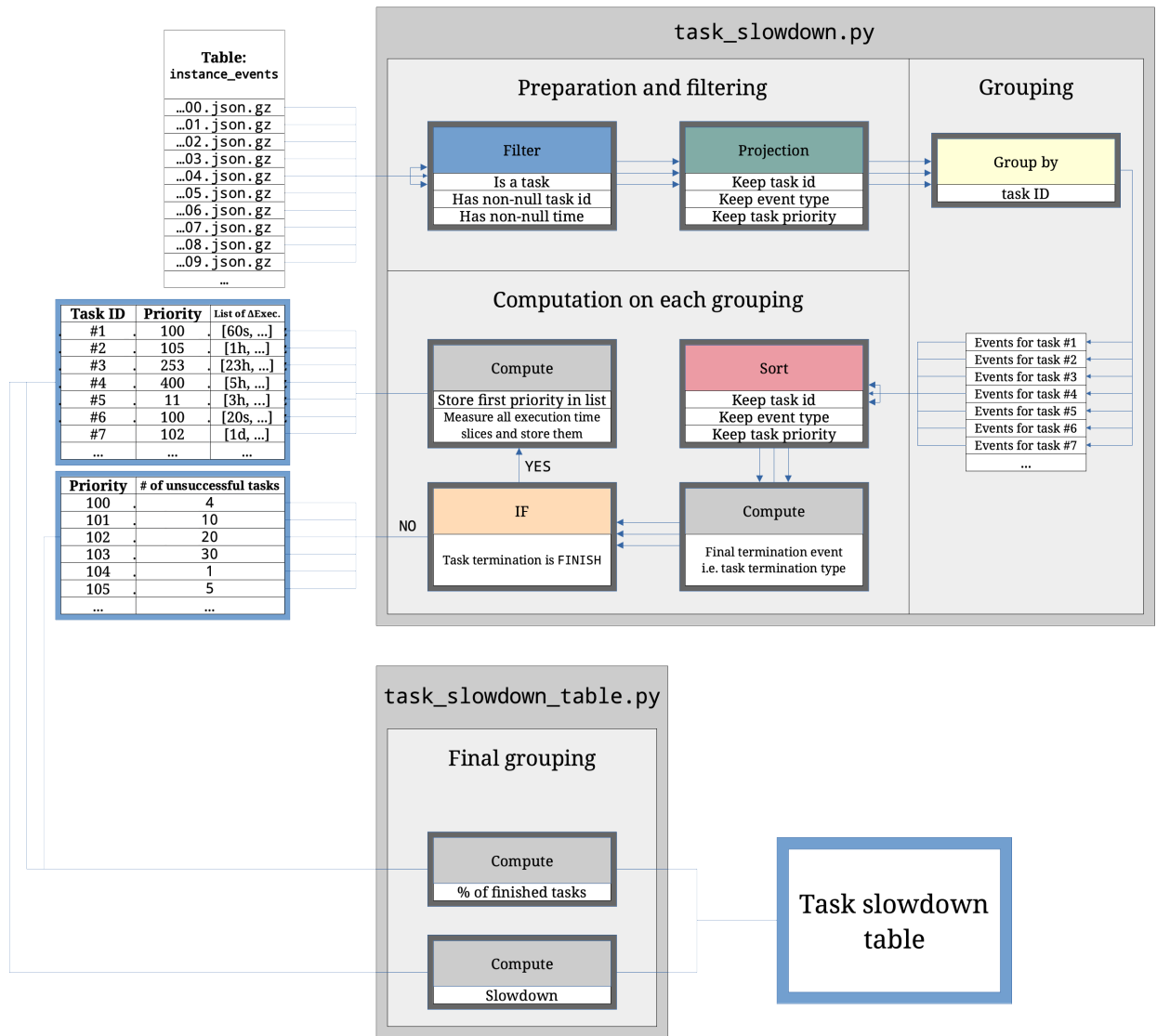
Table:
instance_events

…00.json.gz
…01.json.gz
…02.json.gz
…03.json.gz
…04.json.gz
…05.json.gz
…06.json.gz
…07.json.gz
…08.json.gz
…09.json.gz
…

Task ID	Priority	List of ΔExec.
#1	100	[60s, …]
#2	105	[1h, …]
#3	253	[23h, …]
#4	400	[5h, …]
#5	11	[3h, …]
#6	100	[20s, …]
#7	102	[1d, …]
…	…	…

Priority	# of unsuccessful tasks
100	4
101	10
102	20
103	30
104	1
105	5
…	…

task_slowdown.py

Preparation and filtering

Filter
Is a task
Has non-null task id
Has non-null time

Projection
Keep task id
Keep event type
Keep task priority

Grouping

Group by
task ID

Computation on each grouping

Compute
Store first priority in list
Measure all execution time
slices and store them

Sort
Keep task id
Keep event type
Keep task priority

YES

NO

IF
Task termination is FINISH

Compute
Final termination event
i.e. task termination type

Events for task #1
Events for task #2
Events for task #3
Events for task #4
Events for task #5
Events for task #6
Events for task #7
…

task_slowdown_table.py

Final grouping

Compute
% of finished tasks

Compute
Slowdown

Task slowdown
table

Figure 6. Diagram of the script used for the "task slowdown" query.

5 Analysis: Performance Input of Unsuccessful Executions

Our first investigation focuses on replicating the analysis done by the paper of Rosà et al. paper [2] regarding usage of machine time and resources.

In this section we perform several analyses focusing on how machine time and resources are wasted, by means of a temporal vs. spatial resource analysis from the perspective of single tasks as well as jobs. We then compare the results from the 2019 traces to the ones that were obtained before to understand the workload evolution inside Borg between 2011 and 2019.

We discover that the spatial and temporal impact of unsuccessful executions is very significant, more than in the 2011 traces. In particular, resource usage is overall dominated by tasks with a final KILL termination event.

5.1 Temporal Impact: Machine Time Waste

The goal of this analysis is to understand how much time is spent in doing useless computations by exploring how machine time is distributed over task events and submissions.

Before delving into the analysis itself, we define three kinds of events in a task's lifecycle:

submission: when a task is added or re-added to the Borg system queue, waiting to be scheduled;

scheduling: when a task is removed from the Borg queue and its actual execution of potentially useful computations starts;

termination: when a task terminates its computations either successfully or unsuccessfully.

By partitioning the set of all terminating tasks by their termination event, the analysis aims to measure the total time spent by tasks in 3 different execution phases:

resubmission time: the total of all time intervals between every task termination event and the immediately succeeding task submission event, i.e. the total time spent by tasks waiting to be resubmitted in Borg after a termination;

queue time: the total of all time intervals between every task submission event and the following task scheduling event, i.e. the total time spent by tasks queuing before execution;

running time: the total of all time intervals between every task scheduling event and the following task termination event, i.e. the total time spent by tasks "executing" (i.e. performing potentially useful computations) in the clusters.

In the 2019 traces, an additional "Unknown" measure is counted. This measure collects all the times in which the event transitions between the register events do not allow to safely assume in which execution phase a task may be. Unknown measures are mostly caused by faults and missed event writes in the task event log that was used to generate the traces.

The analysis results are depicted in Figure 7 as a comparison between the 2011 and 2019 traces, aggregating the data from all clusters. Additionally, in Figure 8 cluster-by-cluster breakdown result is provided for the 2019 traces.

The striking difference between 2011 and 2019 data is in the machine time distribution per task termination type. In the 2019 traces, 94.38% of global machine time is spent on tasks that are eventually KILLed. FINISH, EVICT and FAIL tasks respectively register totals of 4.20%, 1.18% and 0.25% machine time.

Considering instead the distribution between execution phase times, the comparison shows very similar behaviour between the two traces, having the "Running" time being dominant (at a total of 16.63% across task terminations in 2019) over the queue and resubmission phases (with respective totals in 2019 of 3.26% and 0.004%).

However, another noteworthy difference between 2011 and 2019 data lies in the new "Unknown" trace dataset present only in the latter traces, registering a total 80.12% of global machine time across al terminations. This data can be interpreted as a strong indication of the "poor quality" of the 2019 traces w.r.t. of accuracy of task event logging.

Considering instead the behaviour of each single cluster in the 2019 traces, no significant difference beween them can be observed. The only notable difference lies between the "Running time"-"Unknown time" ratio in KILLed tasks, which is at its highest in cluster A (at 30.78% by 58.71% of global machine time) and at its lowest in cluster H (at 8.06% by 84.77% of global machine time).

The takeaway from this analysis is that in the 2019 traces a lot of computation time is wasted in the execution of tasks that are eventually KILLed, i.e. unsuccessful.
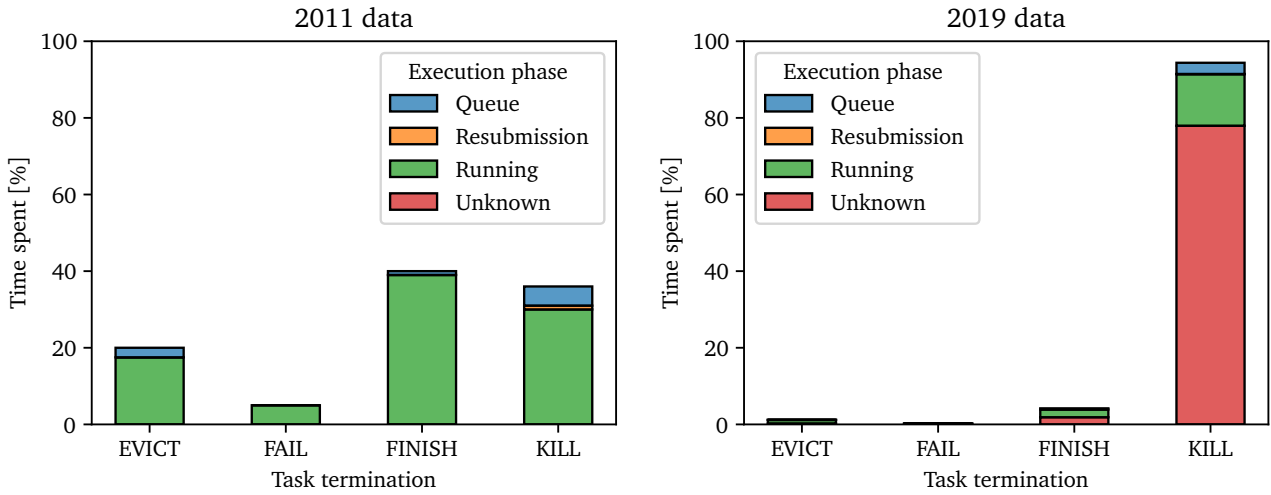
Figure 7. Relative task time spent in each execution phase w.r.t. task termination in 2011 and 2019 (all clusters aggregated) traces. The x-axis shows task termination type, while the y-axis shows total time % spent. Colors break down the time in execution phases. "Unknown" execution times are 2019 specific and correspond to event time transitions that are not consider "typical" by Google.
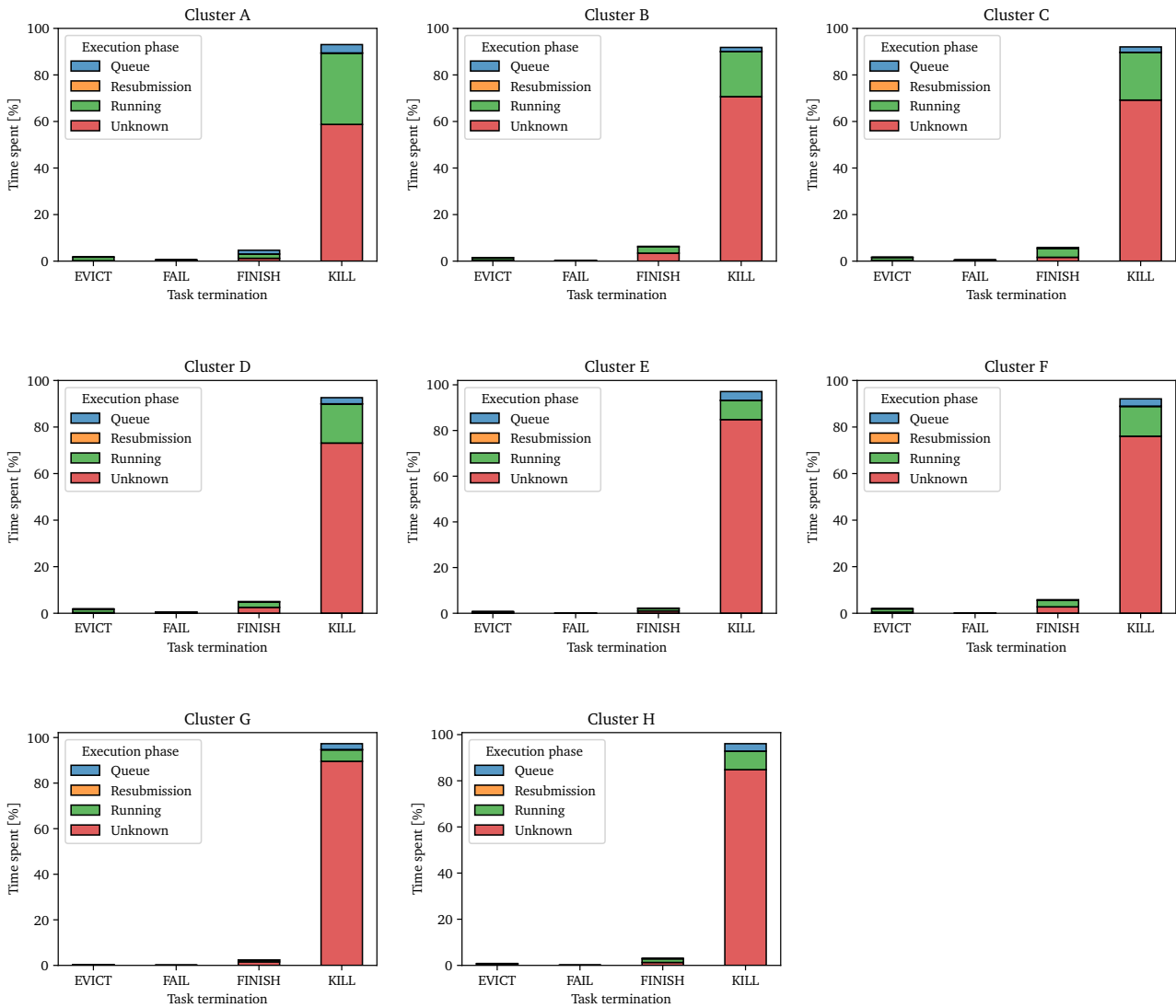
Figure 8. Relative task time spent in each execution phase w.r.t. clusters in the 2019 trace. Refer to Figure 7 for axes description.

5.2 Average Slowdown per Task

This analysis aims to measure the average of an ad-hoc defined parameter we call "slowdown". We define it as the ratio between the total response time across all executions of the task and the response time (i.e. queue time and running time) of the last execution of said task. This metric is especially useful to analyze the impact of unsuccesful executions on each task total execution time w.r.t. the intrinsic workload (i.e. computational time) of tasks.

Refer to Figure 9 for a comparison between the 2011 and 2019 mean task slowdown measures broke down by task priority. Additionally, said means are computed on a cluster-by-cluster basis for 2019 data in Figure 10.

In 2015 Rosà et al. [2] measured mean task slowdown per each task priority value, which at the time were numeric values between 0 and 11. However, in 2019 traces, task priorities are given as a numeric value between 0 and 500. Therefore, to allow an easier comparison, mean task slowdown values are computed by task priority tier over the 2019 data. Priority tiers are semantically relevant priority ranges defined by Tirmazi et al. in 2020 [3] that introduced the 2019 traces. Equivalent priority tiers are also provided next to the 2011 priority values in the table covering the 2015 analysis.

In the given tables, the **% finished** column corresponds to the percentage of FINISHed tasks for that priority or tier. **Mean response [s] (last execution)** instead shows the mean response time of the last task execution of each task in that priority/tier. **Mean response [s] (all executions)** provides a very similar figure, though this column shows the mean response time across all executions. **Mean slowdown** instead provides the mean slowdown value for each task priority/tier.

Comparing the tables in Figure 9 we observe that the maximum mean slowdown measure for 2019 data (i.e. 7.84, for the BEB tier) is almost double of the maximum measure in 2011 data (i.e. 3.39, for priority 3 corresponding to the BEB tier). The "Best effort batch" tier, as the name suggest, is a lower priority tier where failures are more tolerated. Therefore, due to the increased concurrency in the 2019 clusters compared to 2011 and the higher machine time spent for unsuccesful executions (as observed in the previous analysis) and increase slowdown rate for this class is not particularly surprising.

The amount of non-successful task terminations in the 2019 traces is also rather high when compared to 2011 data, as it can evinced by the low percentage of FINISHed tasks across priority tiers.

Another noteworthy difference is in the mean response times for all and last executions: while the mean response is overall shorter in time in the 2019 traces by an order of magnitude, the new traces show an overall significantly higher mean response time than in the 2011 data.

Across 2019 single clusters (as in Figure 10), the data shows a mostly uniform behaviour, other than for some noteworthy mean slowdown spikes. Indeed, cluster A has 82.97 mean slowdown in the "Free" tier, cluster G has 19.06 and 14.57 mean slowdown in the "BEB" and "Production" tier respectively, and Cluster D has 12.04 mean slowdown in its "Free" tier.

5.3 Spatial Impact: Resource Waste

In this analysis we aim to understand how physical resources of machines in the Borg cluster are used to complete tasks. In particular, we compare how CPU and memory resource allocation and usage are distributed among tasks based on their termination type.

Due to limited computational resources w.r.t. the data analysis process, the resource usage for clusters E to H in the 2019 traces is missing. However, a comparison between 2011 resource usage and the aggregated resource usage of clusters A to D in the 2019 traces can be found in Figure 11. Additionally, a cluster-by-cluster breakdown for the 2019 data can be found in Figure 12.

From these figures it is clear that, compared to the relatively even distribution of used resources in the 2011 traces, the distribution of resources in the 2019 Borg clusters became strikingly uneven, registering a combined 86.29% of CPU resource usage and 84.86% memory usage for KILLed tasks. Instead, all other task termination types have a significantly lower resource usage: EVICTed, FAILed and FINISHed tasks register respectively 8.53%, 3.17% and 2.02% CPU usage and 9.03%, 4.45%, and 1.66% memory usage. This resource distribution can also be found in the data from individual clusters in Figure 12, with always more than 80% of resources devoted to KILLed tasks.

Considering now requested resources instead of used ones, a comparison between 2011 and the aggregation of all A-H clusters of the 2019 traces can be found in Figure 13. Additionally, a cluster-by-cluster breakdown for single 2019 clusters can be found in Figure 14.

Here KILLed jobs dominate even more the distribution of resources, reaching a global 97.21% of CPU allocation and

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	Free	67.44%	3598	2939	2.58
2	Best effort batch	90.78%	1835	1782	1.15
3	Best effort batch	95.62%	9683	8294	3.39
4	Best effort batch	78.05%	2006	1890	1.69
5	Best effort batch	100%	58	58	1
6	Best effort batch	77.99%	567	567	1.02
8	Best effort batch	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
Free	42.85%	1374	8	1.15
Best effort batch	11.06%	4139	113	7.84
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 9. 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
Free	0.33%	5769	1203	82.97
Best effort batch	212.62%	71108	14201	5.17
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
Free	45.21%	12047	5588	1.18
Best effort batch	71.84%	1018454	550288	8.47
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
Free	73.36%	172214	5553	1.12
Best effort batch	52.96%	1236666	997117	7.40
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
Free	42.82%	22831	5506	1.15
Best effort batch	50.56%	1154060	1135023	12.04
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
Free	48.15%	33050	40073	1.44
Best effort batch	0.47%	280811	205838	8.06
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
Free	45.86%	187447	37069	1.09
Best effort batch	44.29%	1368306	1563086	6.14
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
Free	33.85%	64718	15473	1.14
Best effort batch	104.33%	294959	184724	19.06
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
Free	28.79%	310534	290058	1.12
Best effort batch	107.03%	947368	527812	7.33
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 10. Mean task slowdown for each cluster and each task priority for single clusters in the 2019 traces. Refer to Figure 9 for a legend of the columns.
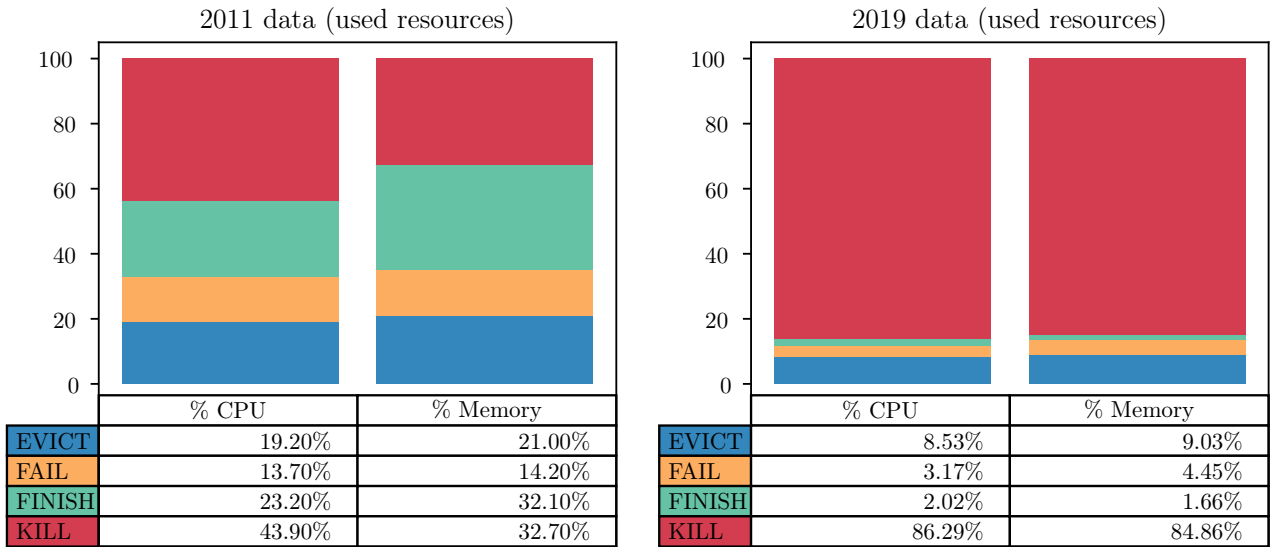
Figure 11. Percentages of CPU and RAM resources used by tasks w.r.t. task termination type in 2011 and 2019 traces (total of clusters A to D). The x axis is the type of resource, y-axis is the percentage of resource used and color represents task termination. Numeric values are displayed below the graph as a table.
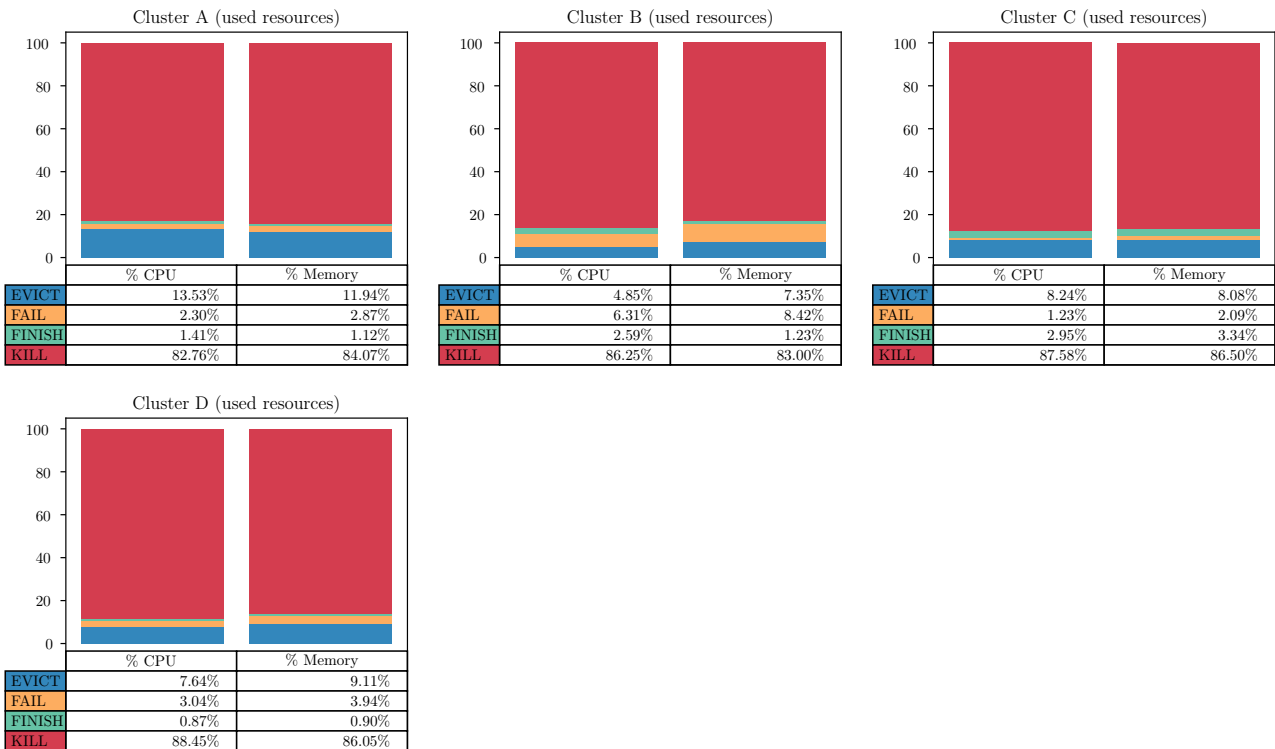
Figure 12. Percentages of CPU and RAM resources used by tasks w.r.t. task termination type for clusters A to D in 2019 traces. Refer to figure 11 for plot explaination.

14
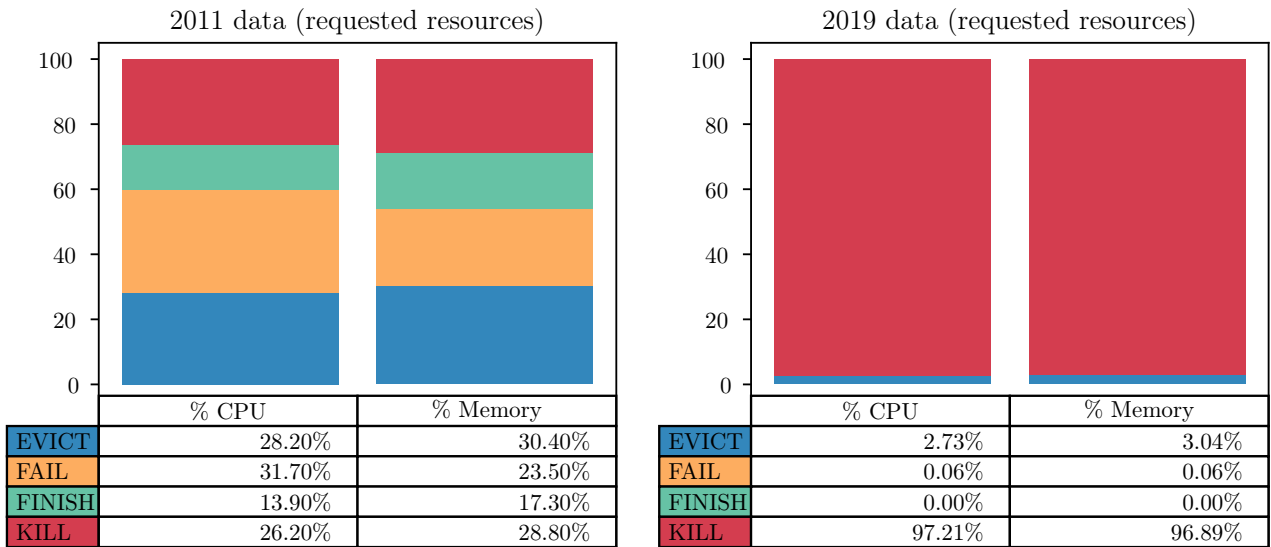
Figure 13. Percentages of CPU and RAM resources requested by tasks w.r.t. task termination type in 2011 and 2019 traces. The x axis is the type of resource, y-axis is the percentage of resource used and color represents task termination. Numeric values are displayed below the graph as a table.

Figure 14. Percentages of CPU and RAM resources requested by tasks w.r.t. task termination type for in 2019 traces. Refer to figure 13 for plot explaination.

a global 96.89% of memory allocation. Even in allocations, the KILL lead is followed by (in order) EVICTed, FAILed and FINISHed jobs, with respective CPU allocation figures of 2.73%, 0.06% and 0.0012% and memory allocation figures of 3.04%, 0.06% and 0.012%.

Behaviour across clusters (as evinced in Figure 14) in terms of requested resources is pretty homogeneous, with the exception of cluster A having a relatively high 2.85% CPU and 3.42% memory resource requests from EVICTed tasks and cluster E having a noteworthy 1.67% CPU and 1.31% memory resource resquests from FINISHed tasks.

With more than 98% of both CPU and memory resources used by (and more than 99.99% of both CPU and memory resources requested by) non-successful tasks, it is clear the spatial resource waste is high in the 2019 traces.

6 Analysis: Patterns of Task and Job Events

This section aims to use some of the tecniques used in section IV of the Rosà et al. paper [2] to find patterns and interpendencies between task and job events by gathering event statistics at those events. In particular, Section 6.1 explores how the success of a task is inter-correlated with its own event patterns, which Section 6.2 explores even further by computing task success probabilities based on the number of task termination events of a specific type. Finally, Section 6.3 aims to find similar correlations, but at the job level.

The results found the the 2019 traces seldomly show the same patterns in terms of task events and job/task distributions, in particular highlighting again the overall non-trivial impact of KILL events, no matter the task and job termination type.

6.1 Unsuccessful Task Event Patterns

In this analysis we compute the distribution of termination events by type at the task-level events, namely EVICT, FAIL, FINISH and KILL termination events.

A comparison of the termination event distribution between the 2011 and 2019 traces is shown in Figure 15. Additionally, a cluster-by-cluster breakdown of the same data for the 2019 traces is shown in Figure 16.

Each table from these figures shows the mean and the 95-th percentile of the number of termination events per task, broke down by task termination. In addition, the table shows the mean number of EVICT, FAIL, FINISH, and KILL for each task event termination.

The first observation we make is that the mean number of events per EVICTed and FAILed tasks increased more than 5-fold (namely from 2.372 to 78.710 and from 3.130 to 24.962 respectively). Also observing the 95-th percentile we can say that the number of events per task has generally increased overall.

As observed in 2011, 2019 Borg tasks have all a multitude of events with different types, with FINISHed tasks experiencing almost always FINISH events and unsuccessful tasks and the same observation holding for KILLed tasks and their KILL events. Differently from the 2011 data, EVICTed tasks seem to experience an high number of KILL events as well (25.795 on average per task, over 78.710 overall events on average). A similar phenomena can be observed with KILLed jobs and their EVICT events (1.876 on average per task with a 8.763 event overall average).

Considering cluster-by-cluster behaviour in the 2019 traces (as reported in Figure 16) the general observations still hold for each cluster, albeit with event count averages having different magnitudes. Notably, cluster E registers the highest per-event average, with FAILed tasks experiencing 111.471 FAIL events out of 112.384.

6.2 Conditional Probability of Task Success

In this analysis we measure the conditional probability of task success given a number of specific unsuccessful (i.e. EVICT, FAIL and KILL) events. This analysis was conducted to better understand how a given number of unsuccessful events could affect the termination of the task it belongs to.

Conditional probabilities of each unsuccessful event type are shown in the form of a plot in Figure 17, comparing the 2011 traces with the overall data from the 2019 ones, and in Figure 18, as a cluster-by-cluster breakdown of the same data for the 2019 traces.

In Figure 17 the 2011 and 2019 plots differ in their x-axis: for 2011 data conditional probabilities are computed for a maximum event count of 30, while for 2019 data are computed for up to 50 events of a specific kind. Nevertheless, another quite striking difference between the two plots can be seen: while 2011 data has relatively smooth decreasing curves for all event types, the curves in the 2019 data almost immediately plateau with no significant change easily observed after 5 events of any kind.

The presence of even one KILL event almost surely causes the corresponding task to terminate in an unsuccessful way: a task with no KILL events has 97.16% probability of success, but tasks with 1 to 5 KILL events have 0.02%, 0.20%, 0.44%, 0.04%, and 0.07% probabilities of success respectively. The same effect can be observed, albeit in a less drastic fashion, for the EVICT and FAIL curves. The EVICT curve has for tasks with 0 to 5 kill events 19.70%, 15.94%, 1.94%, 1.67%, 0.35% and 0.00% success probabilities repectively. The FAIL probability curve has instead 18.55%, 1.79%, 14.49%, 2.08%, 2.40%, and 1.29% success probabilities for the same range.

Considering cluster-to-cluster behaviour in the 2019 traces (as shown in Figure 18), some clusters show quite similar behaviour to the aggregated plot (namely clusters A, F, and H), while some other clusters show very oscillating probability distribution function curves for EVICT and FINISH curves. KILL behaviour is instead homogeneous even

Task termination	Mean number of events Overall (95th 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 (95th p.)	EVICT	FAIL	FINISH	KILL
EVICT	78.710 (342)	52.242	0.673	0	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 15. 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 (95th p.)	EVICT	FAIL	FINISH	KILL
EVICT	103.228 (719)	73.694	0.769	0	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 (95th p.)	EVICT	FAIL	FINISH	KILL
EVICT	83.018 (394)	64.817	0.240	0	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 (95th p.)	EVICT	FAIL	FINISH	KILL
EVICT	98.437 (444)	73.716	1.813	0	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 (95th p.)	EVICT	FAIL	FINISH	KILL
EVICT	76.759 (366)	62.001	0.700	0	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 (95th p.)	EVICT	FAIL	FINISH	KILL
EVICT	17.678 (72)	11.781	0.106	0	5.791
FAIL	112.384 (28)	0.458	111.471	0	0.456
FINISH	2.029 (2)	0.014	0.008	1.999	0.008
KILL	13.505 (64)	1.288	0.057	0	12.160

(e) Cluster E

Task termination	Mean number of events Overall (95th p.)	EVICT	FAIL	FINISH	KILL
EVICT	70.146 (114)	23.974	0.192	0	45.980
FAIL	41.087 (54)	0.279	39.257	0	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 (95th p.)	EVICT	FAIL	FINISH	KILL
EVICT	136.032 (490)	77.429	0.303	0	58.299
FAIL	8.948 (8)	0.016	8.593	0	0.339
FINISH	14.176 (2)	0.015	0.002	14.154	0.005
KILL	32.320 (164)	6.909	0.135	0	25.276

(g) Cluster G

Task termination	Mean number of events Overall (95th p.)	EVICT	FAIL	FINISH	KILL
EVICT	14.734 (40)	6.733	0.837	0	7.165
FAIL	41.067 (120)	0.600	37.600	0	2.867
FINISH	3.681 (2)	0.024	0.014	3.633	0.011
KILL	17.976 (98)	0.633	0.170	0	17.173

(h) Cluster H

Figure 16. 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.

18
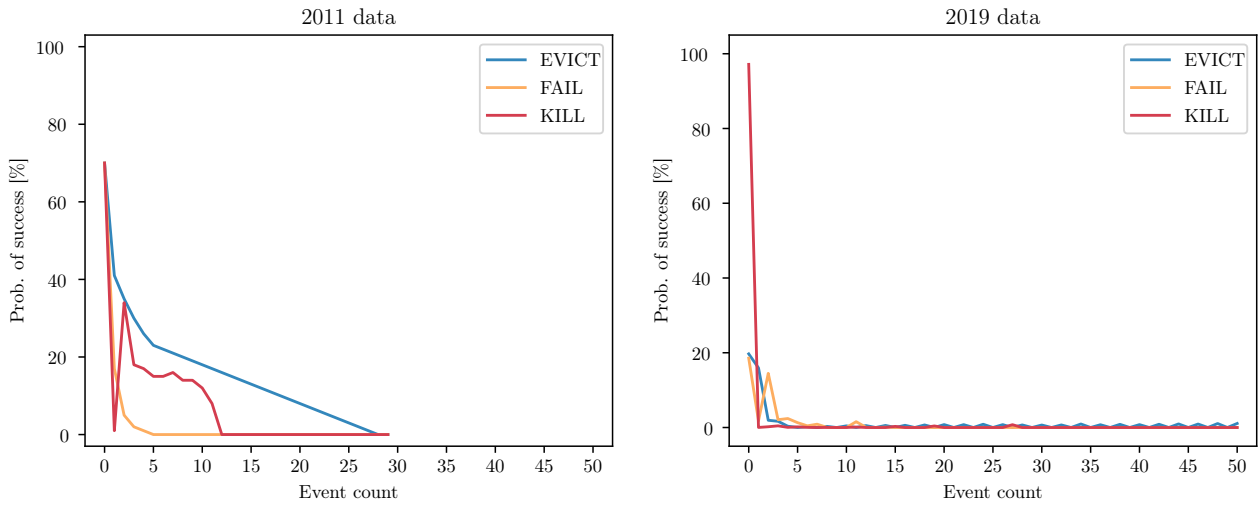
Figure 17. Conditional probability of task success given a number of specific unsuccessful events observed, i.e., EVICT, FAIL and KILL for 2011 and 2019 (all clusters aggregated) traces. For 2011 data the probability was computed for a maximum event count of 30, while in 2019 it was computed for up to 50 events of a specific kind.
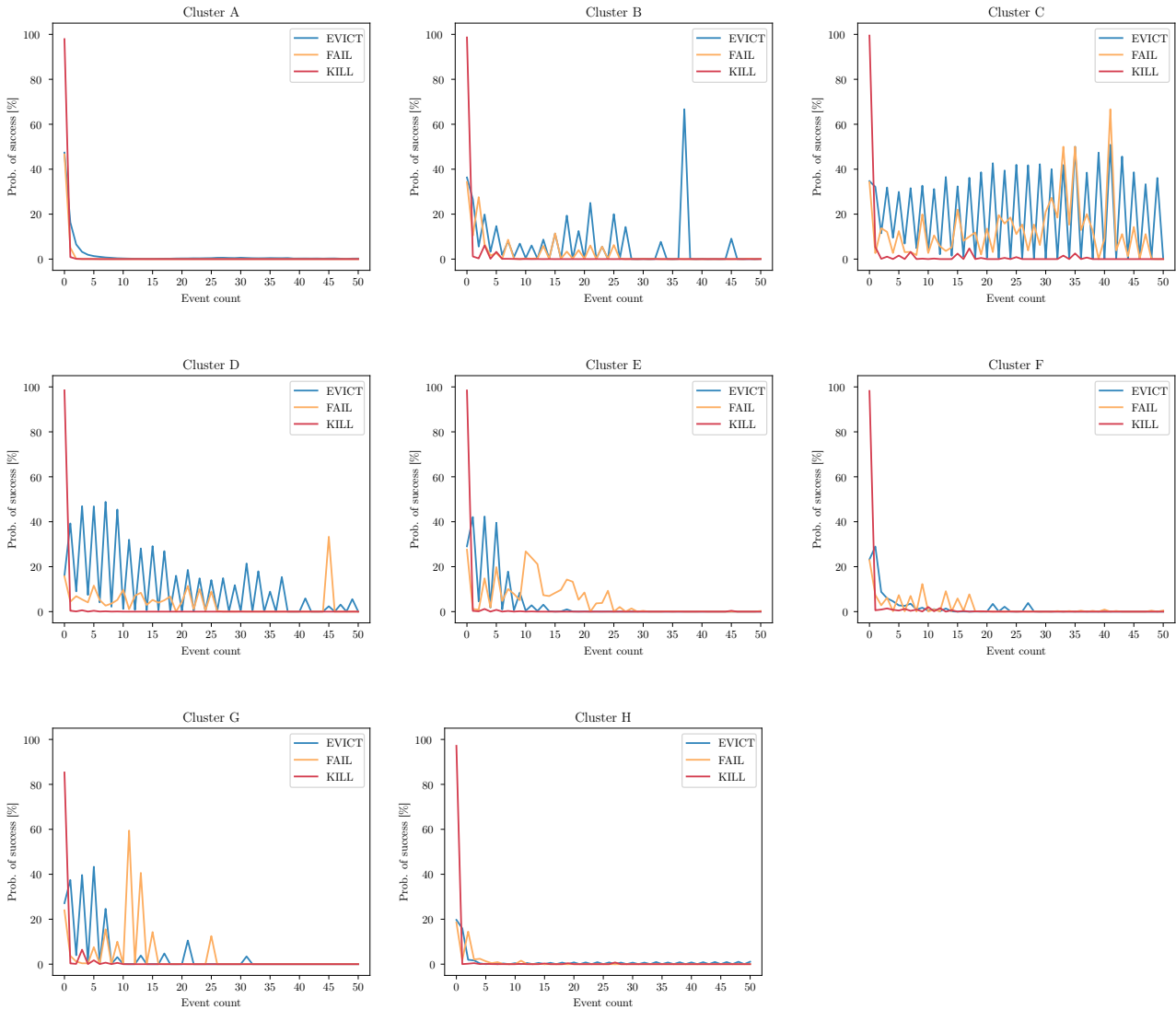
Figure 18. Conditional probability of task success given a number of specific unsuccessful events observed, i.e., EVICT, FAIL and KILL for each cluster in the 2019 traces.

19

on a single cluster basis.

6.3 Unsuccessful Job Event Patterns

The analysis uses very similar techniques to the ones used in Section 6.1, but focusing at the job level instead. The aim is to better understand the task-job level relationship and to understand how task-level termination events can influence the termination state of a job.

A comparison of the analyzed parameters between the 2011 and 2019 traces is shown in Figure 19. Additionally, a cluster-by-cluster breakdown of the same data for the 2019 traces is shown in Figure 20.

Considering the distribution of number of tasks in a job, the 2019 traces show a decrease for the mean figure (e.g. for FAILed jobs, with a mean 60.5 tasks per job in 2011 and a mean 43.126 tasks per job in 2019) and a fluctuation of the 95-th percentile figure (e.g. for FAILed jobs it rose from 110 to 200, but for KILLed job the figure decreased from 400 to 178).

Considering the distribution of the number of task-wise termination events instead, the 2019 traces show values generally one or two orders of magnitude below the ones in 2011. While the behaviour of EVICTed jobs stays the same, FAILed and KILLed jobs show a dramatic difference in the event distribution, with KILL becoming the most popular event task-wise with mean 12.833 and 11.337 task events per job respectively. Finally, the FINISHed job category has a new event distribution too, with FINISH task events being the most popular at 1.778 events per job in the 2019 traces.

The cluster-by-cluster comparison in Figure 20 shows that the number of tasks per job are generally distributed similarly to the aggregated data, with only cluster H having remarkably low mean and 95-th percentiles overall. Event-wise, for EVICTed, FINISHed, and KILLed jobs again the distributions are similar to the aggregated one. For some clusters (namely B, C, and D), the mean number of FAIL and KILL task events for FINISHed jobs is almost the same. Additionally, it is noteworthy that cluster A has no EVICTed jobs.

Job termination	# of tasks mean. (95th p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1 (1)	1	0	0	0
FAIL	60.5 (110)	139.0	788.5	49.2	9.5
FINISH	2.7 (1)	0.4	0.1	$5 \cdot 10^{-4}$	2.7
KILL	86.8 (400)	13.3	20.9	26.9	62.7

(a) 2011 data

Job termination	# of tasks mean. (95th p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0	0	0
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 19. Mean number of tasks and event distribution per job type for between 2011 and 2019 (all clusters aggregated) traces. The tables show and mean and 95-th percentile for the number of tasks in a job, and additionally show the mean of job-wise total of task termination events.

Job termination	# of tasks mean. (95th 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. (95th p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0	0	0
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. (95th p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.001	0	0	0
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. (95th p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0	0	0
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. (95th p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0	0	0
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. (95th p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0	0	0
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. (95th p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0	0	0
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. (95th p)	Mean number of events			
		EVICT	FAIL	FINISH	KILL
EVICT	1.000 (1)	1.000	0	0	0
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 20. Mean number of tasks and event distribution per job type for each cluster in the 2019 traces. The tables show and mean and 95-th percentile for the number of tasks in a job, and additionally show the mean of job-wise total of task termination events.

7 Analysis: Potential Causes of Unsuccessful Executions

This section re-applies the tecniques used in Section V of the Rosà et al. paper [2] to find causes for unsuccessful events related to task-level parameters (analyzed in Section 7.1), usage of machine resources by tasks (analyzed in Section 7.2), and job-level parameters (analyzed in Section 7.3). In all the analyses we use the "event rate" metric, which represents the relative percentage of termination type events over a certain task/job parameter configuration. We compute this metric for all the possible terminations (i.e. EVICT, FAIL, FINISH and KILL) in order to find correlations with the several trace parameters.

7.1 Task Event Rates vs. Task Priority, Event Execution Time, and Machine Concurrency.

This analysis shows event rates (i.e. the relative percentage of termination type events) over different configurations of task-level parameters. Figure 21 and Figure 22 show the distribution of event rates over the various task priority tiers. Figure 23 and Figure 24 show the distribution of event rates over the total event execution time. Finally, Figure 25 and Figure 26 show the distribution of event rates over the metric of machine concurrency, defined as the number of co-executing tasks on the machine and at the moment the termination event is recorded.

From this analysis we can make the following observations:

- The behaviour of the curves in the task priority distributions (in Figure 21 and Figure 22) for the 2019 traces is almost the opposite of the 2011 ones, i.e. in-between priorities have higher kill rates while priorities at the extremum have lower kill rates;

- The event execution time curves (in Figure 23 and Figure 24) for the 2019 traces are quite different than 2011 ones, here it seems there is a good correlation between short task execution times and finish event rates, instead of the "U shape" curve found in the Rosà et al. 2015 DSN paper [2];

- The behaviour among different clusters for the event execution time distributions in Figure 24 seem quite uniform;

- The machine concurrency metric, for which a distribution of event rates is computed in Figure 25 and Figure 26, seems to play little role in the event termination distribution, as for all concurrency factors the KILL event rate is around 90% with little fluctuation.

7.2 Task Event Rates vs. Requested Resources, Resource Reservation, and Resource Utilization

This analysis is concerned with the distribution of event rates over several resources related parameters. Figure 27 and Figure 28 show the distribution of task event rates w.r.t. the amount of CPU the task has requested, while Figure 29 and Figure 30 show task events rates vs. requested memory. Figure 31 and Figure 32 show the distribution of task event rates w.r.t. the amount of CPU that has collectively requested on the machine where the task is running, while Figure 33 and Figure 34 show a similar distribution but for memory. Finally Figure 35 and Figure 36 show the distribution of task event rates w.r.t. the amount of CPU the task has really been utilized, while Figure 37 and Figure 38 show task events rates vs. used memory.

From this analysis we can make the following observations:

- In the 2019 trace, the amount of requested CPU resources seem to play little effect on the job termination as it can evinced in Figure 27 and Figure 28. Instead, the job rate distributions w.r.t. the amount of requested memory (Figure 29 and Figure 30) show no discernable pattern;

- Overall a significant increment in the killed event rate can be observed. They seem to dominate all event rates measures;

- Among all clusters in Figure 28 there can be noted the dominance of the killed event rate. In 2011, it was observed a more dominant behaviour by the success event rate curve;

- For each analysed distribution, clusters do not show a common behaviour of the curves. Some are similar, but they are generally distinguishable;

- In Figure 35 there can be seen that while a drastic decrease of the killed event rate curve is observed as the CPU utilization increases, the success event rate does not increase much.
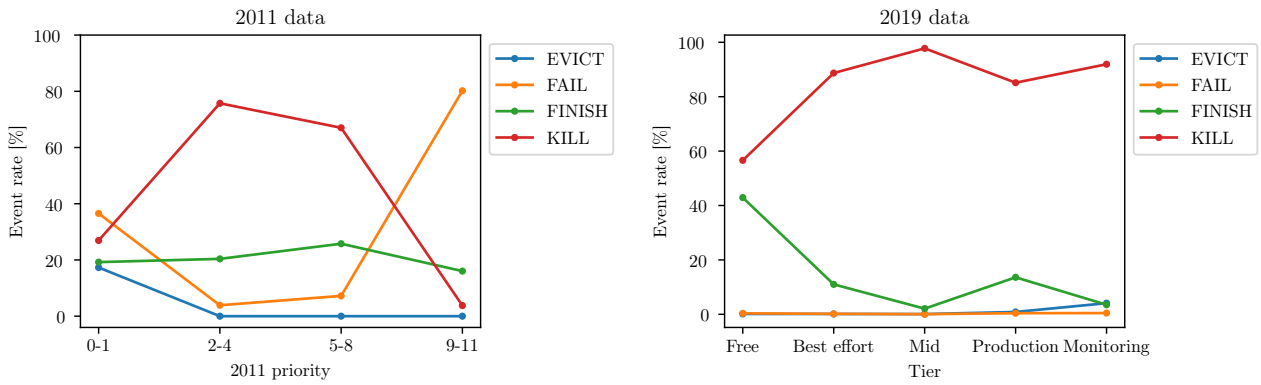
Figure 21. Task event rates vs. task priority and task termination in 2011 and 2019 (all clusters aggregated) traces. For 2019 traces tier classes instead of raw priority values are shown: 2011's [0, 1] priority range corresponds to the "Free" tier, range [2, 8] corresponds to the "Best effort batch" tier, range [9−10] corresponds to the "Production" tier and priority 11 corresponds to the "Monitoring" tier.
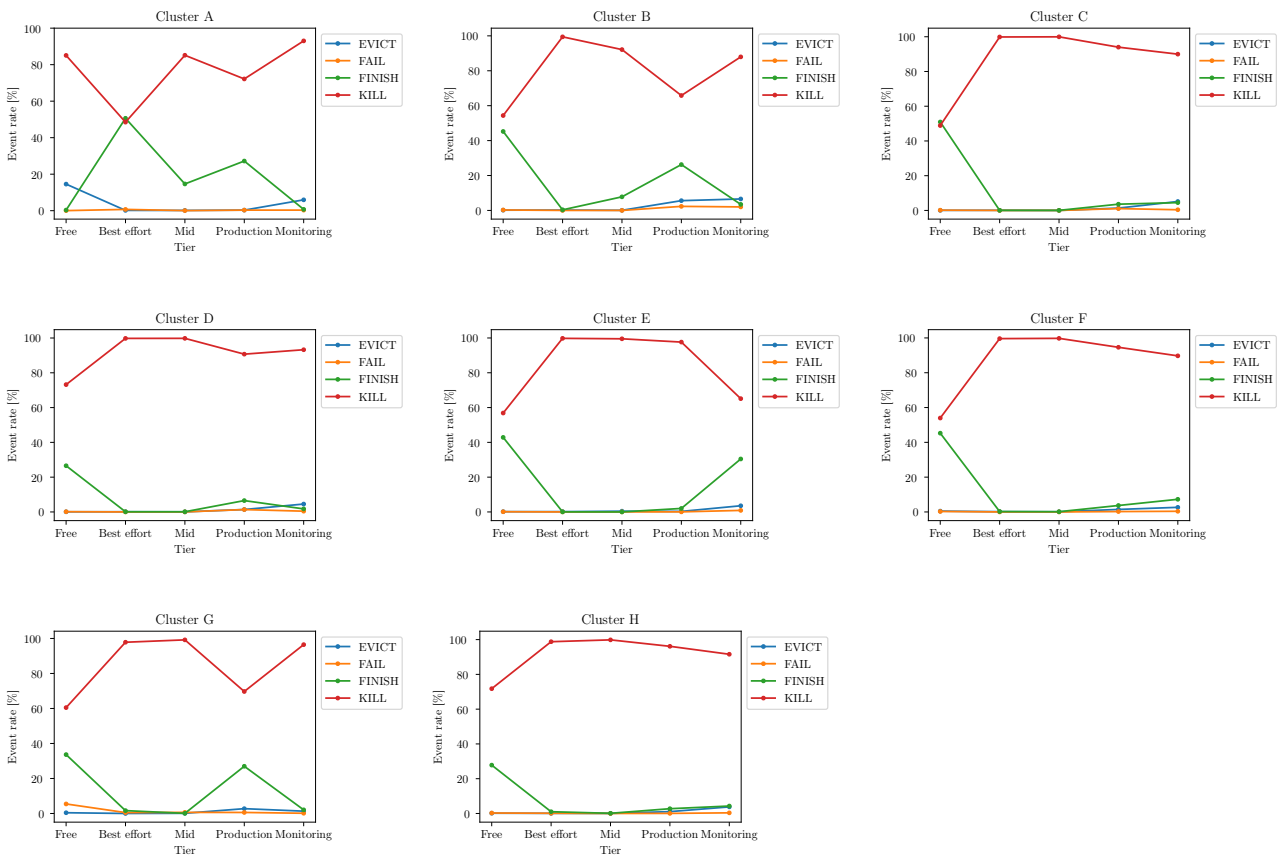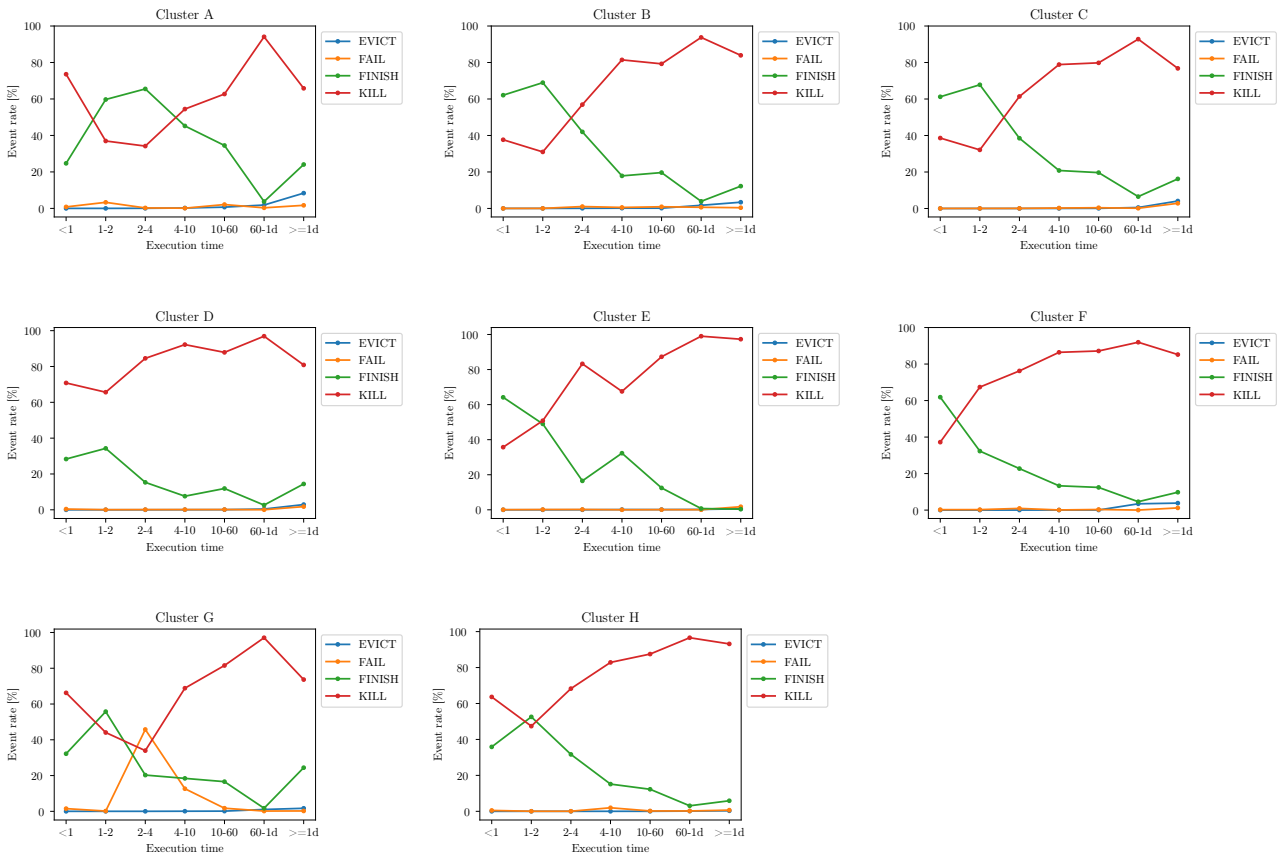

Figure 22. Task event rates vs. task priority tier and final task termination for each cluster in the 2019 traces.
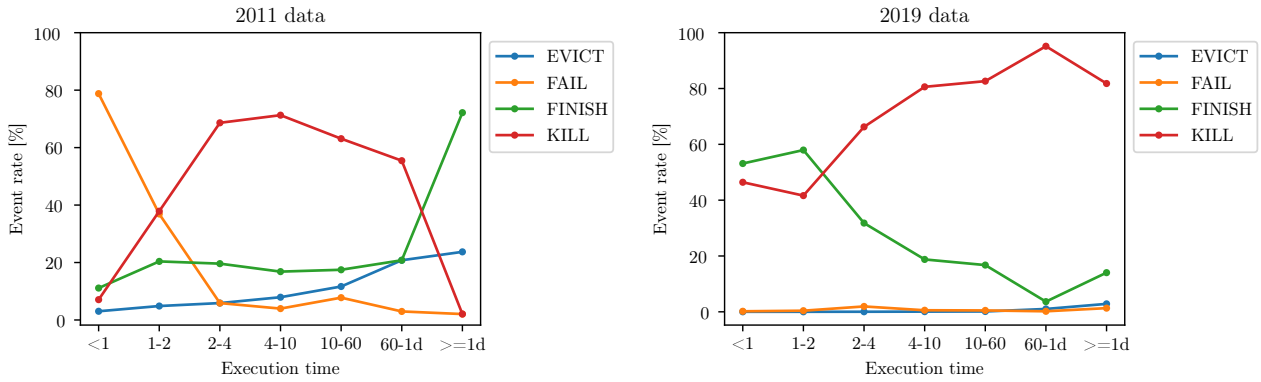
Figure 23. Task event rates vs. event execution time and final task termination in 2011 and 2019 (all clusters aggregated) traces. Execution time classes are defined in minutes, with the exception of "1d" which means one day.

Figure 24. Task event rates vs. event execution time and final task termination for each cluster in the 2019 traces. Refer to figure 23 for interpretation of the execution time classes.
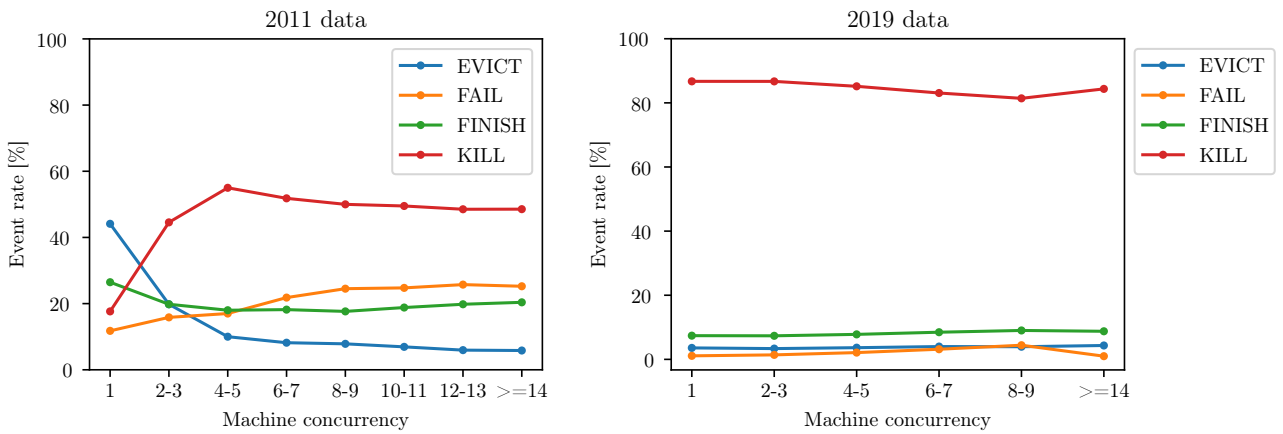
Figure 25. Task event rates vs. machine concurrency and final task termination in 2011 and 2019 (all clusters aggregated) traces. Machine concurrency is defined as the number of co-executed tasks on the machine where the analyzed task is executing.
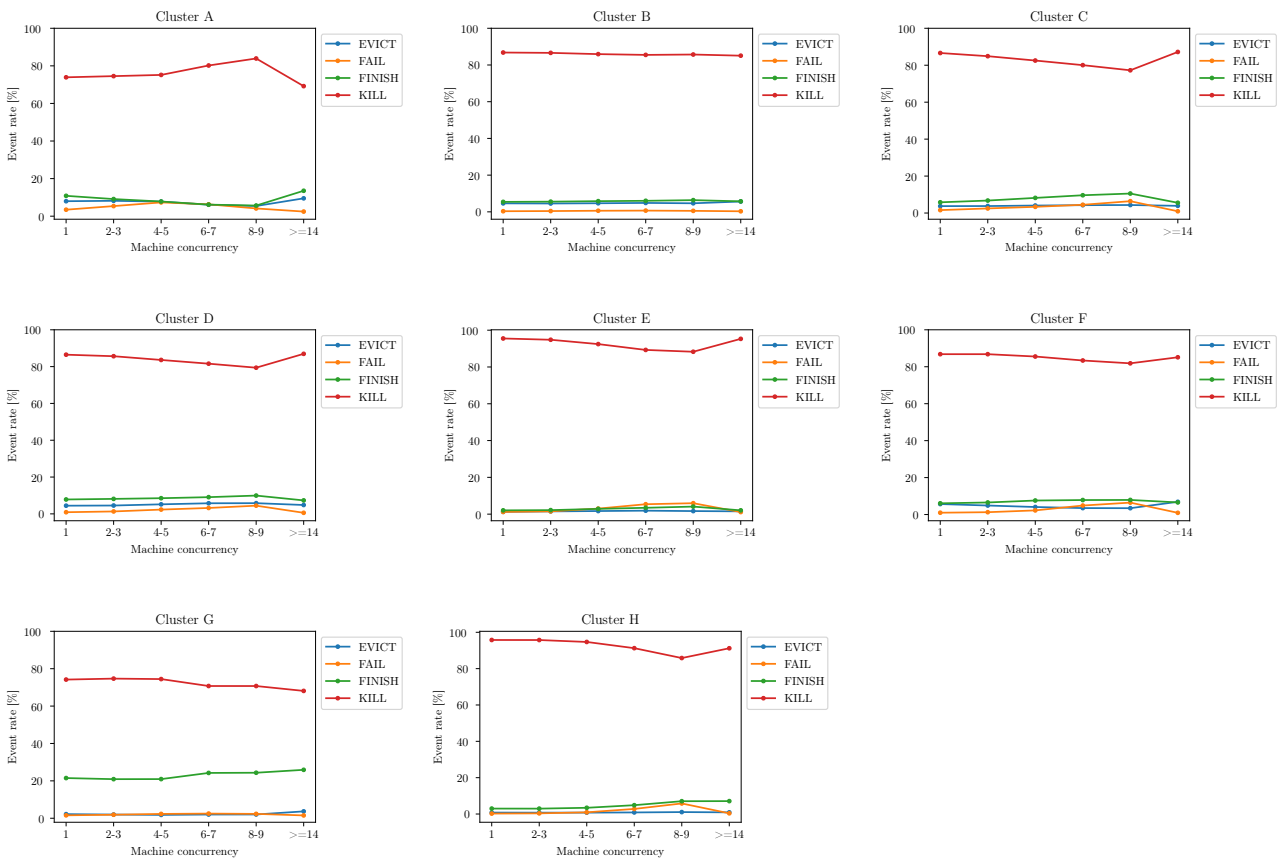
Figure 26. Task event rates vs. machine concurrency and final task termination in each 2019 cluster. Refer to figure 25 for the definition of "machine concurrency".
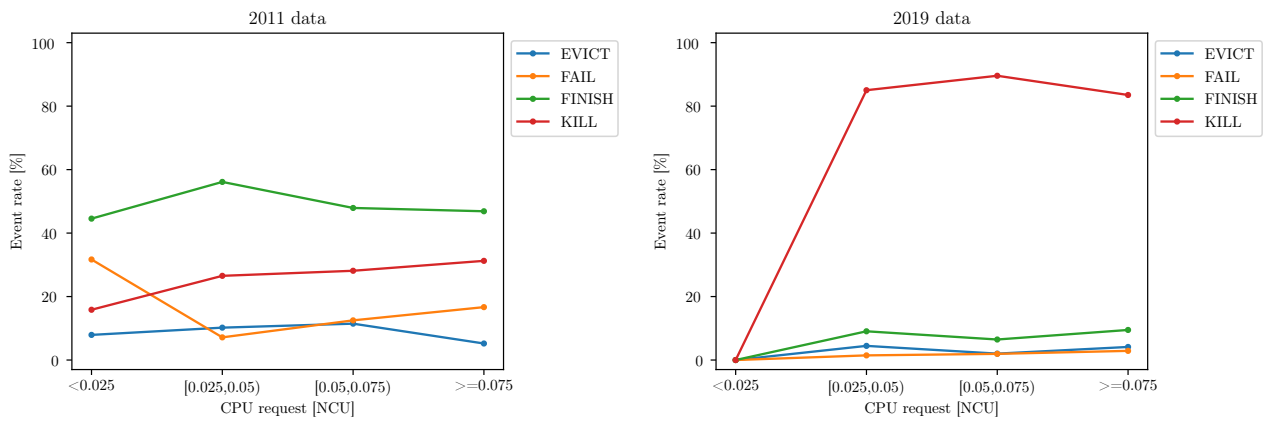
Figure 27. Task event rates vs. requested CPU (expressed in *NCUs*), w.r.t. task termination for 2011 and 2019 (all cluster aggregated) traces.
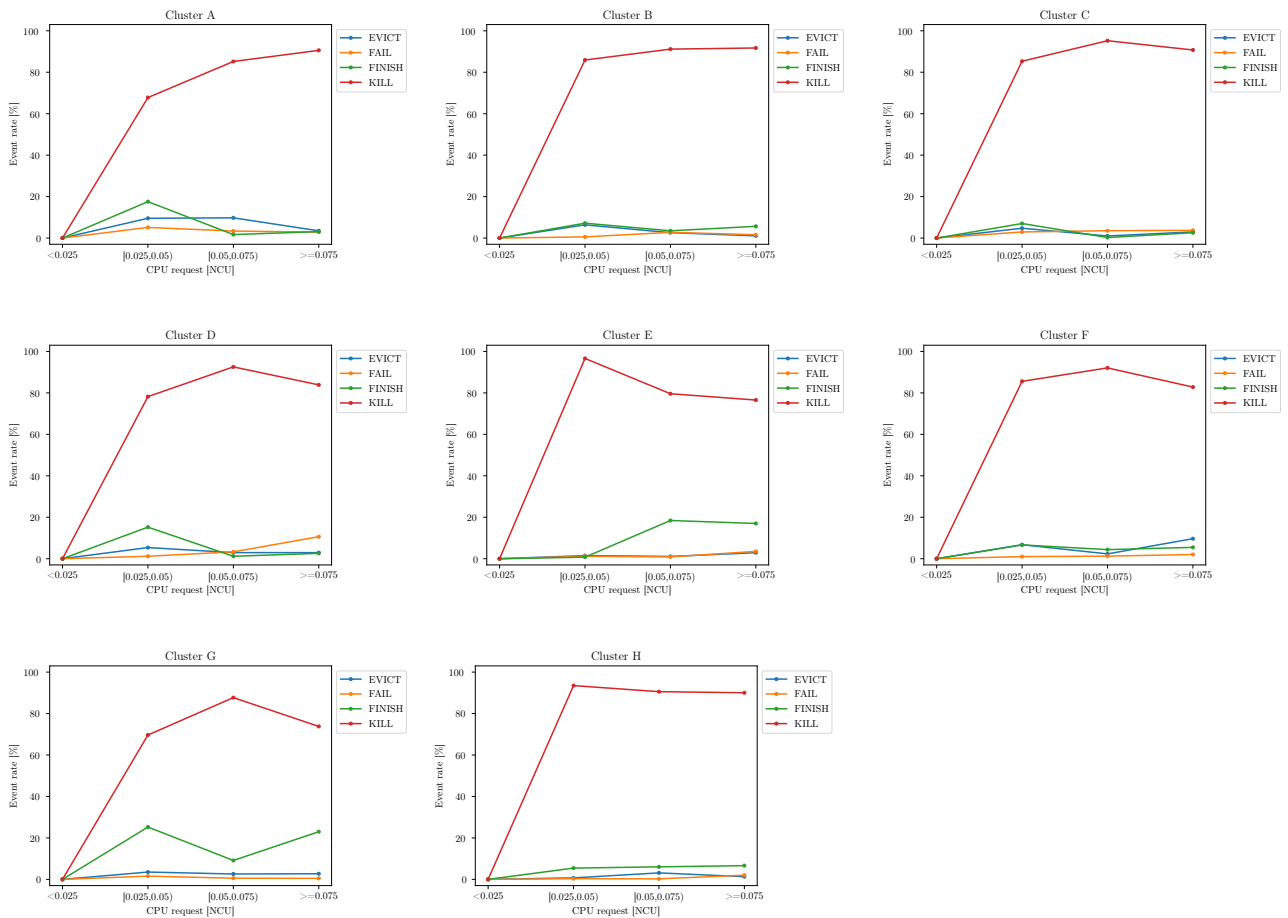
Figure 28. Task event rates vs. requested CPU (expressed in *NCUs*), w.r.t. task termination for each cluster in the 2019 traces.
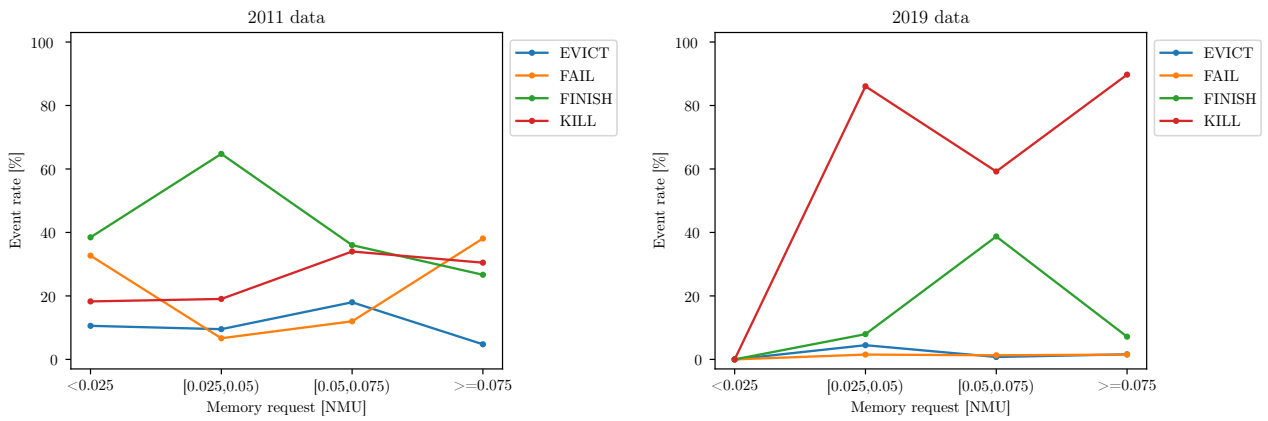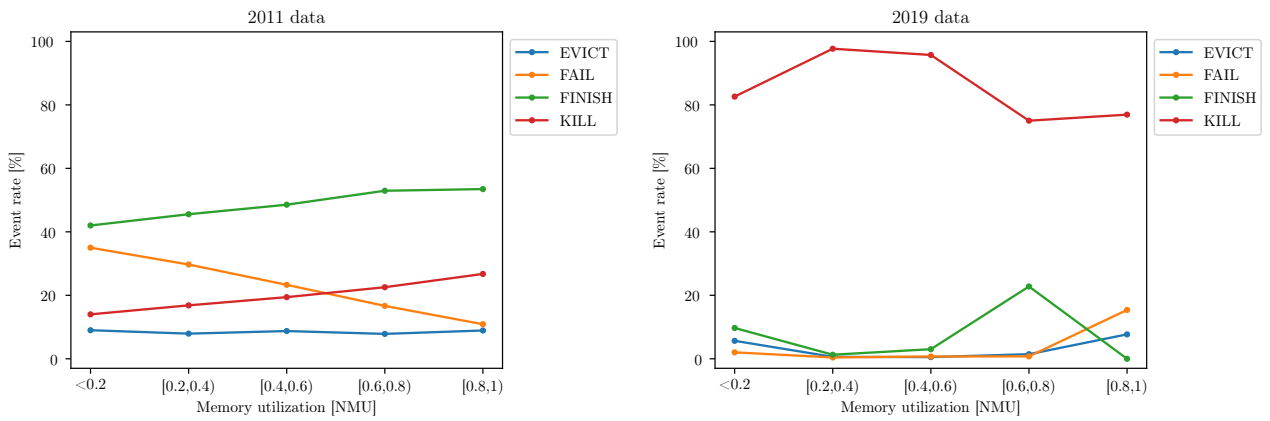
Figure 29. Task event rates vs. requested memory (expressed in *NMUs*), w.r.t. task termination for 2011 and 2019 (all cluster aggregated) traces.
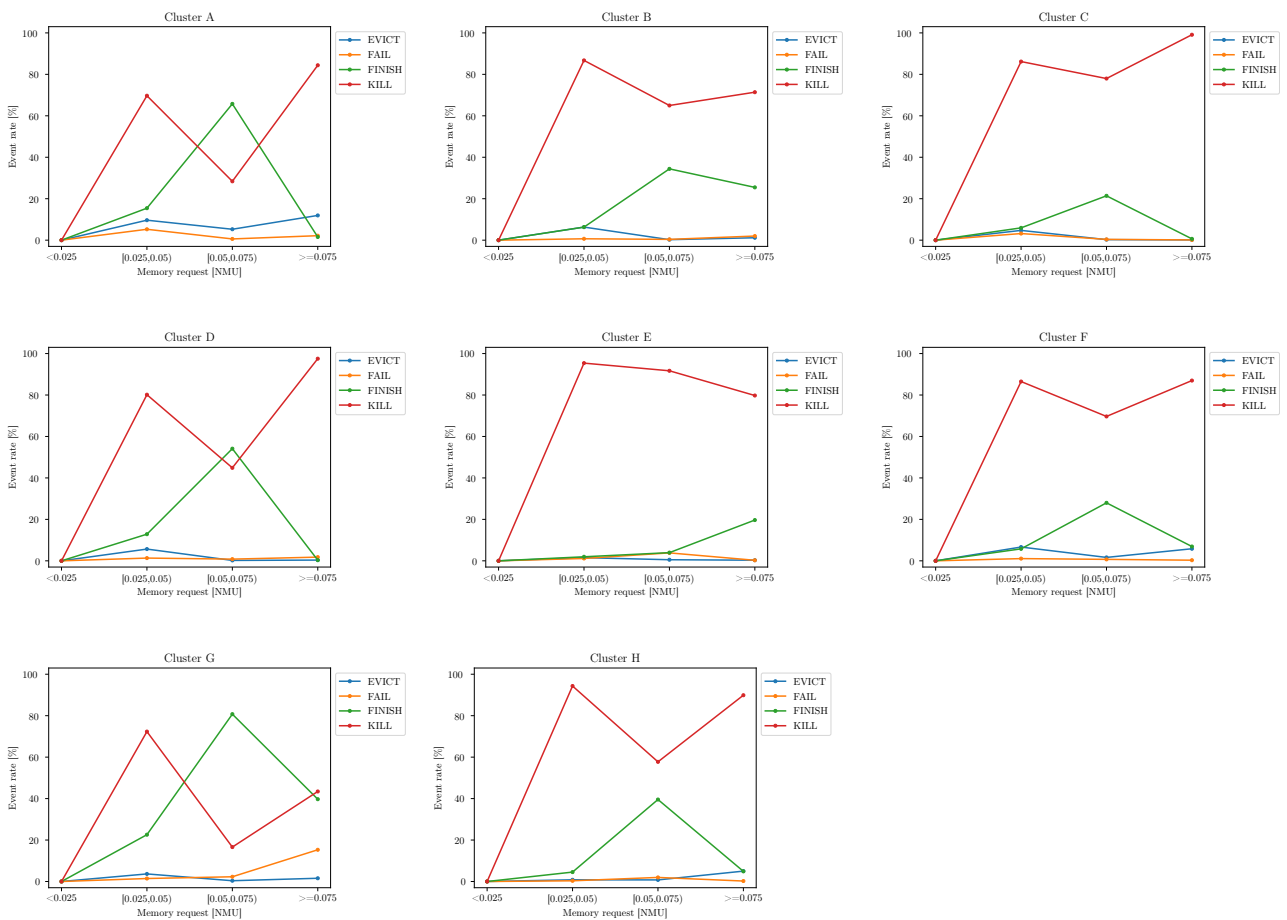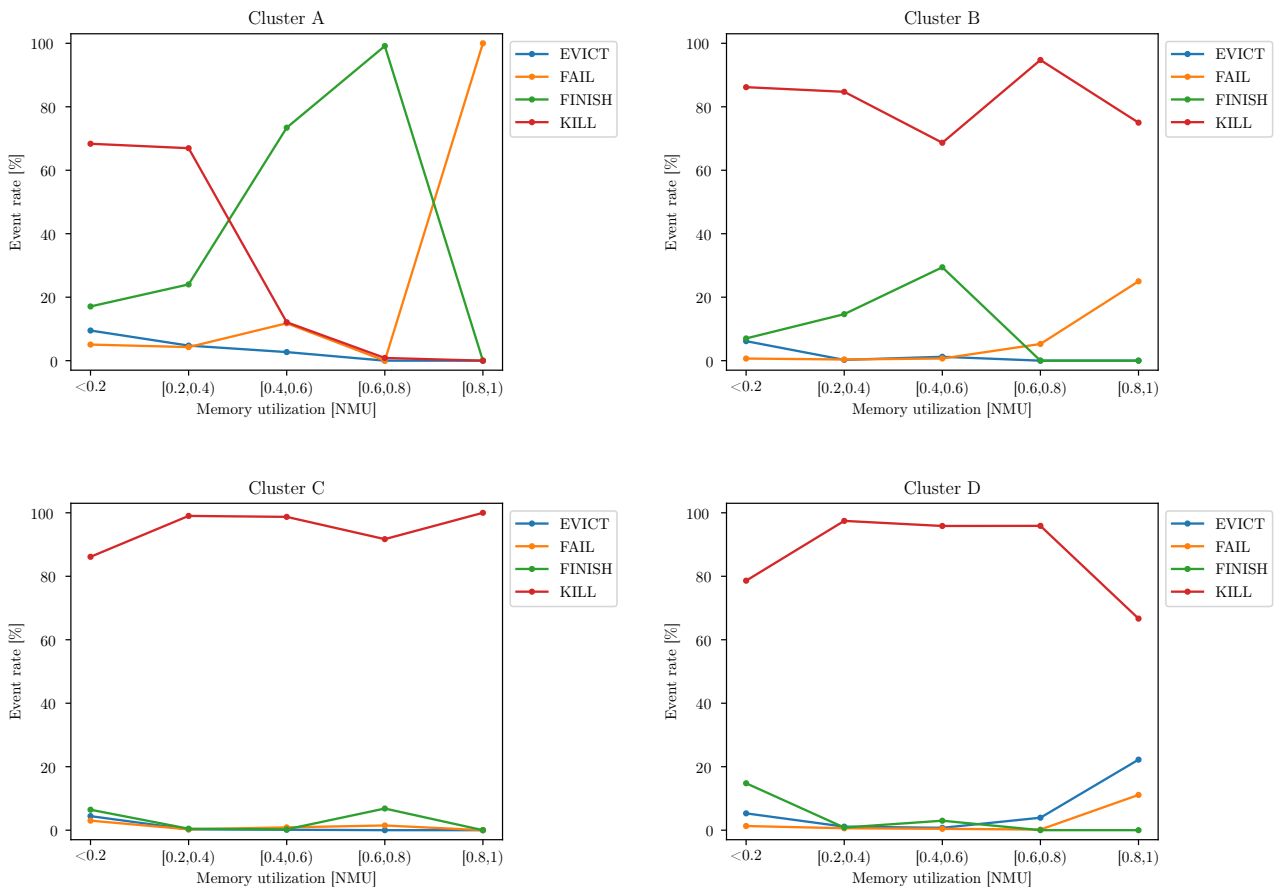

Figure 30. Task event rates vs. requested memory (expressed in *NMUs*), w.r.t. task termination for each cluster in the 2019 traces.
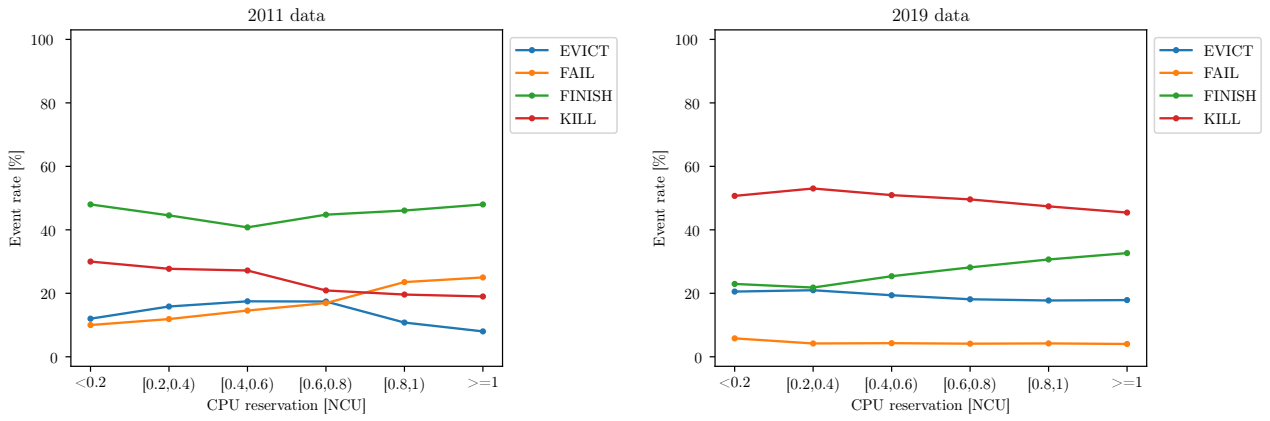
Figure 31. Task event rates vs. reserved CPU (expressed in *NCUs*), w.r.t. task termination for 2011 and 2019 (clusters A,B,E,F aggregated) traces.
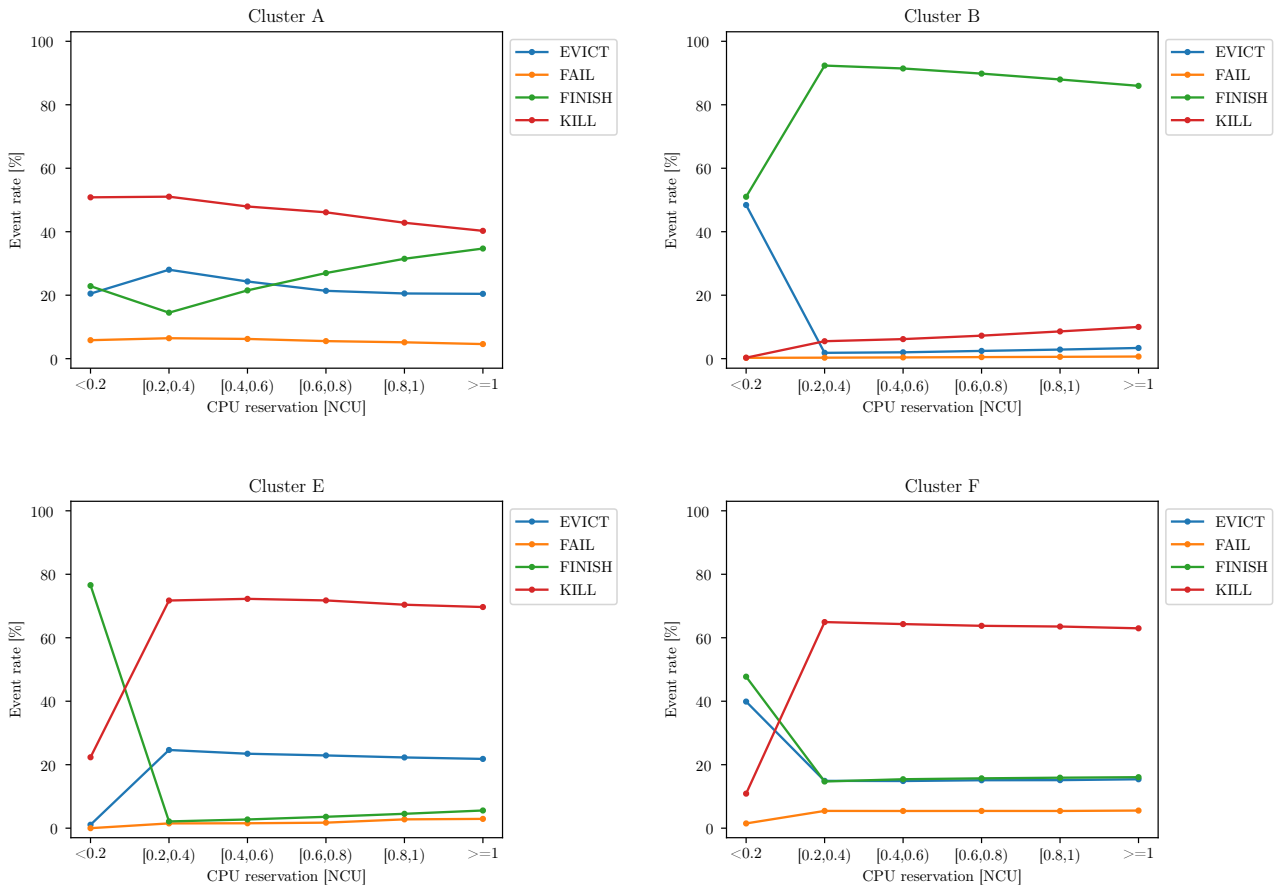
Figure 32. Task event rates vs. reserved CPU (expressed in *NCUs*), w.r.t. task termination for clusters A,B,E,F in the 2019 traces.
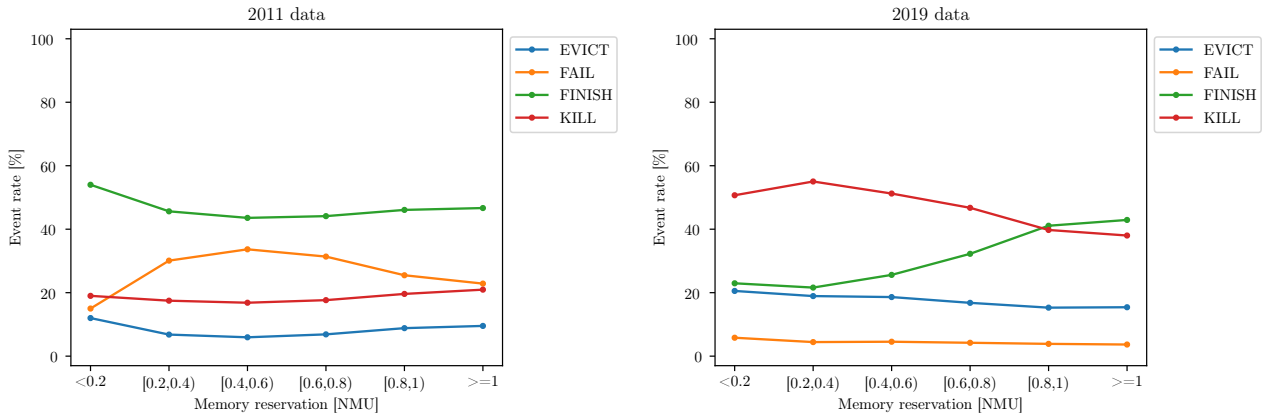
Figure 33. Task event rates vs. reserved memory (expressed in *NMUs*), w.r.t. task termination for 2011 and 2019 (clusters A,B,E,F aggregated) traces.
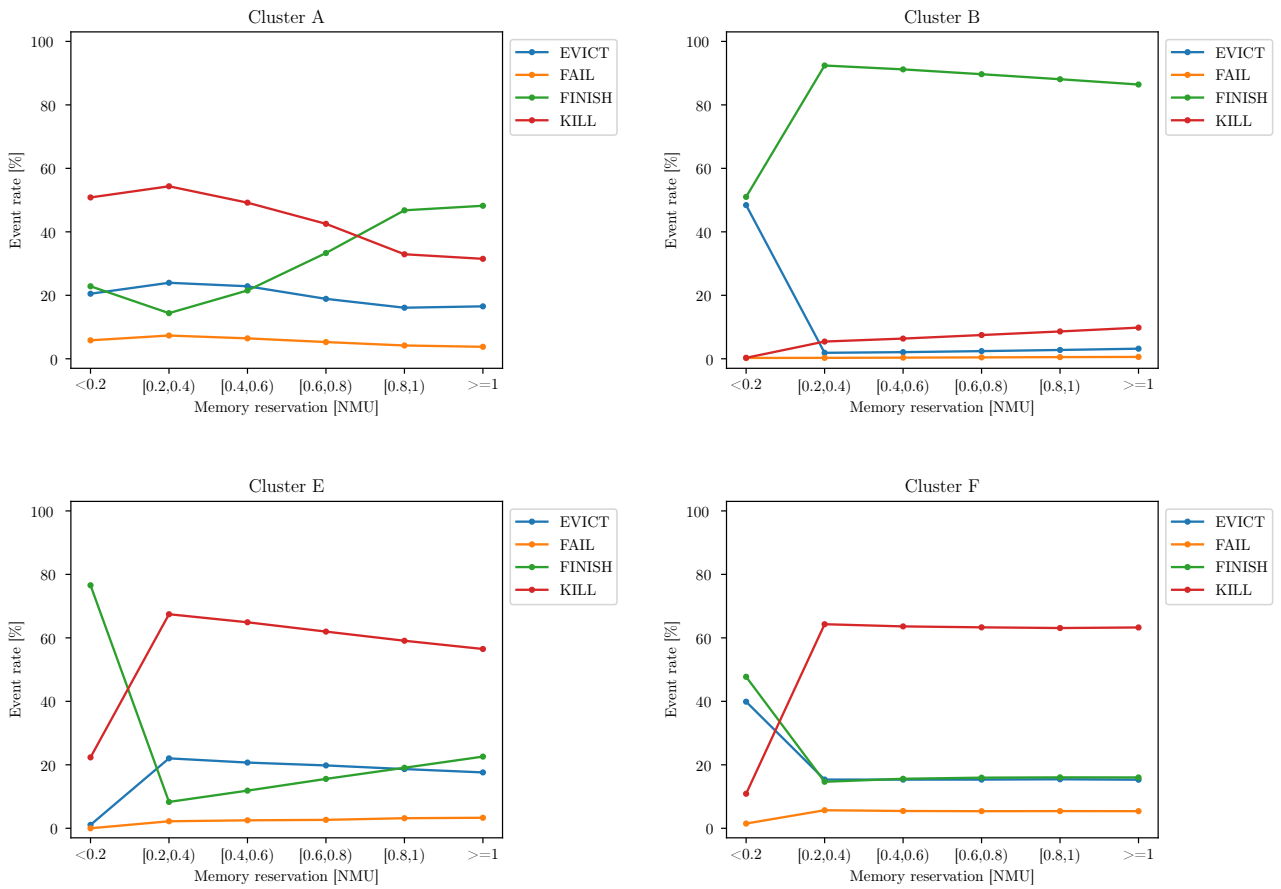
Figure 34. Task event rates vs. reserved memory (expressed in *NMUs*), w.r.t. task termination for clusters A,B,E,F in the 2019 traces.
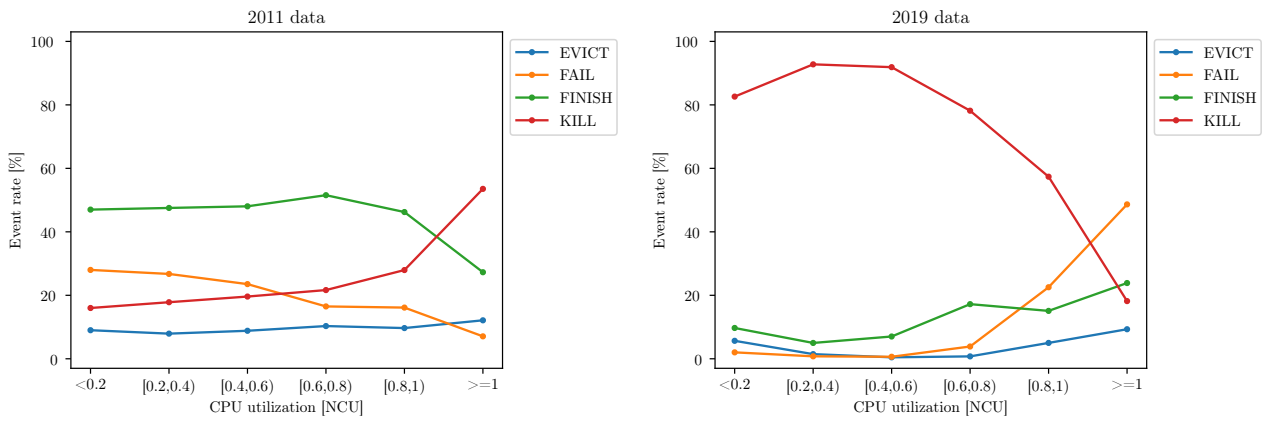
Figure 35. Task event rates vs. used CPU (expressed in *NCUs*), w.r.t. task termination for 2011 and 2019 (clusters A-D aggregated) traces.
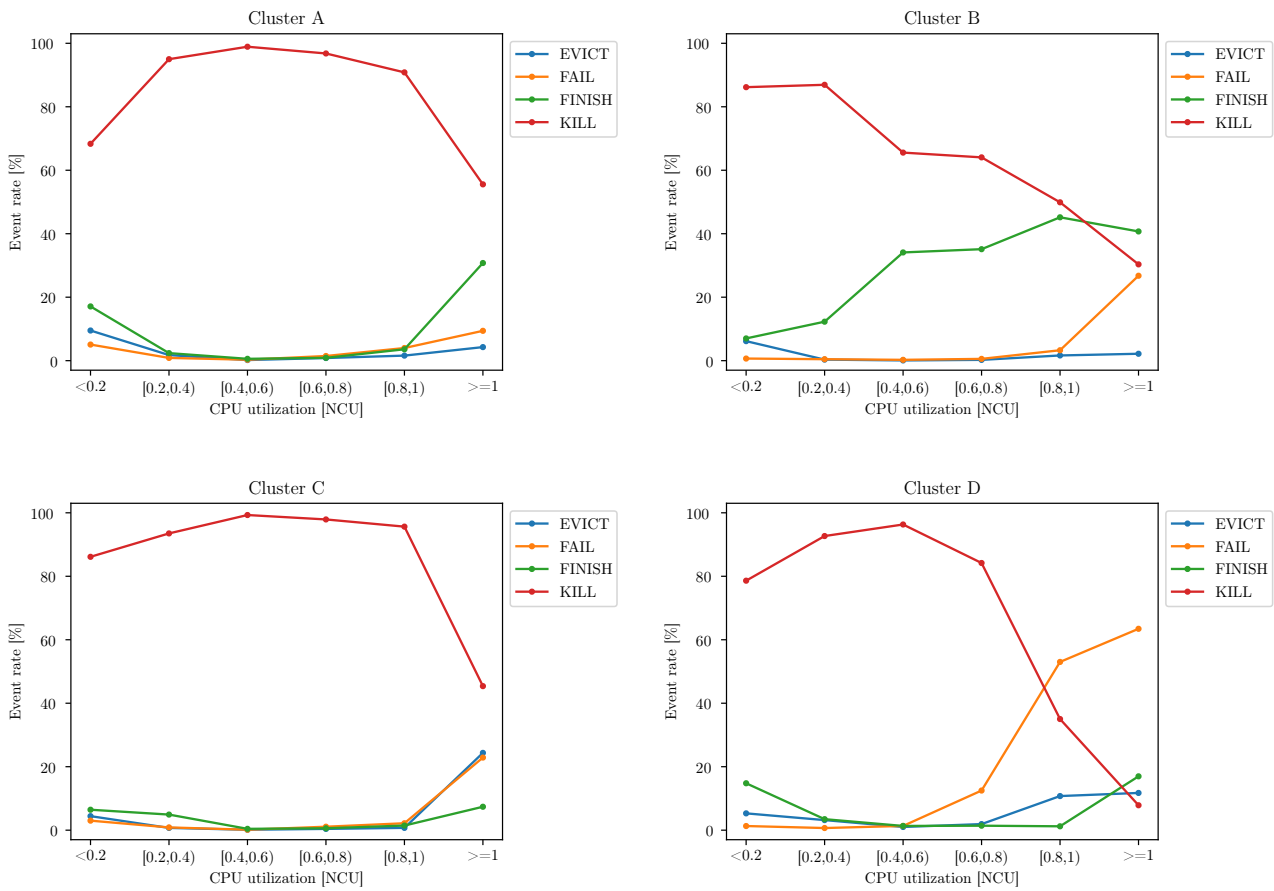
Figure 36. Task event rates vs. used CPU (expressed in *NMUs*), w.r.t. task termination for clusters A-D in the 2019 traces.

Figure 37. Task event rates vs. used memory (expressed in *NMUs*), w.r.t. task termination for 2011 and 2019 (clusters A-D aggregated) traces.

Figure 38. Task event rates vs. used memory (expressed in *NMUs*), w.r.t. task termination for clusters A-D in the 2019 traces.

7.3   Job Event Rates vs. Job Size, Job Execution Time, and Machine Locality

This analysis shows job event rates (i.e. the relative percentage of termination type events) over different configurations of job size, job execution time and machine locality.

Figure 39 and Figure 40 provide the plots of job event rates versus the job size. Job size is defined as the number of tasks belonging to the job. Figure 41 and Figure 42 provide the plots of the job event rates versus execution time. Figure 43 and Figure 44 provide the plots of the job event rates versus machine locality. Machine locality is defined as the ratio between the number of machines used to execute the tasks inside the job and the job size.

By analysing these plots, we can make the following observations:

- There can be noted significant variations in the behaviour of the curves between clusters;

- 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. As can be seen in Figure 39, an higher number of tasks (i.e., an higher job size) seems to be correlated to an higher killed event rate in 2019 rather than in 2011. In Figure 41, we observe the best success event rate for a job execution time of 4-10 minutes, while in 2011, it seemed that the finish event rate increases along with the job execution time;

- There still seems to be a strong correlation between short execution job times and successful final termination, and likewise for kills and higher job terminations. Especially for these two curves, in most cases also between the clusters, their behaviour suggests a specular trend;

- As can be seen in Figure 43, across all clusters, a machine locality factor of 1 seems to lead to the highest success event rate, while in 2011 the same machine locality factor led to the lowest success event rate.
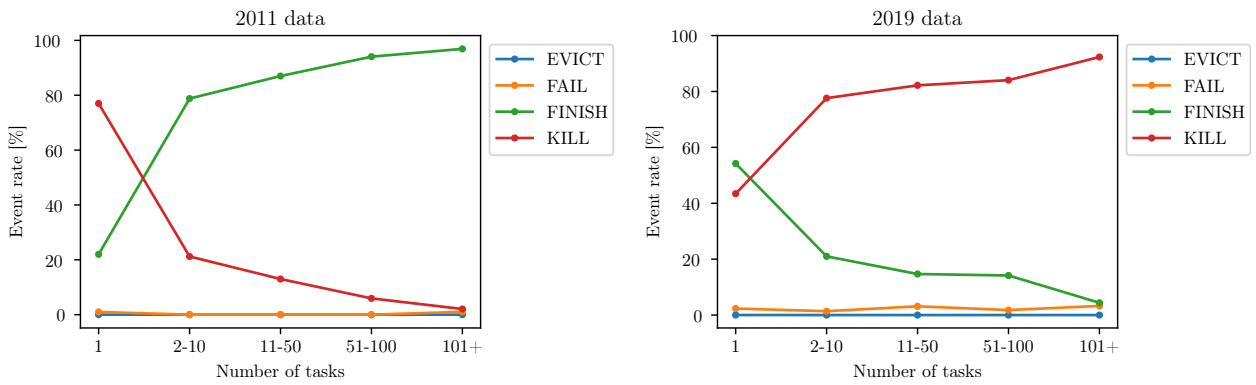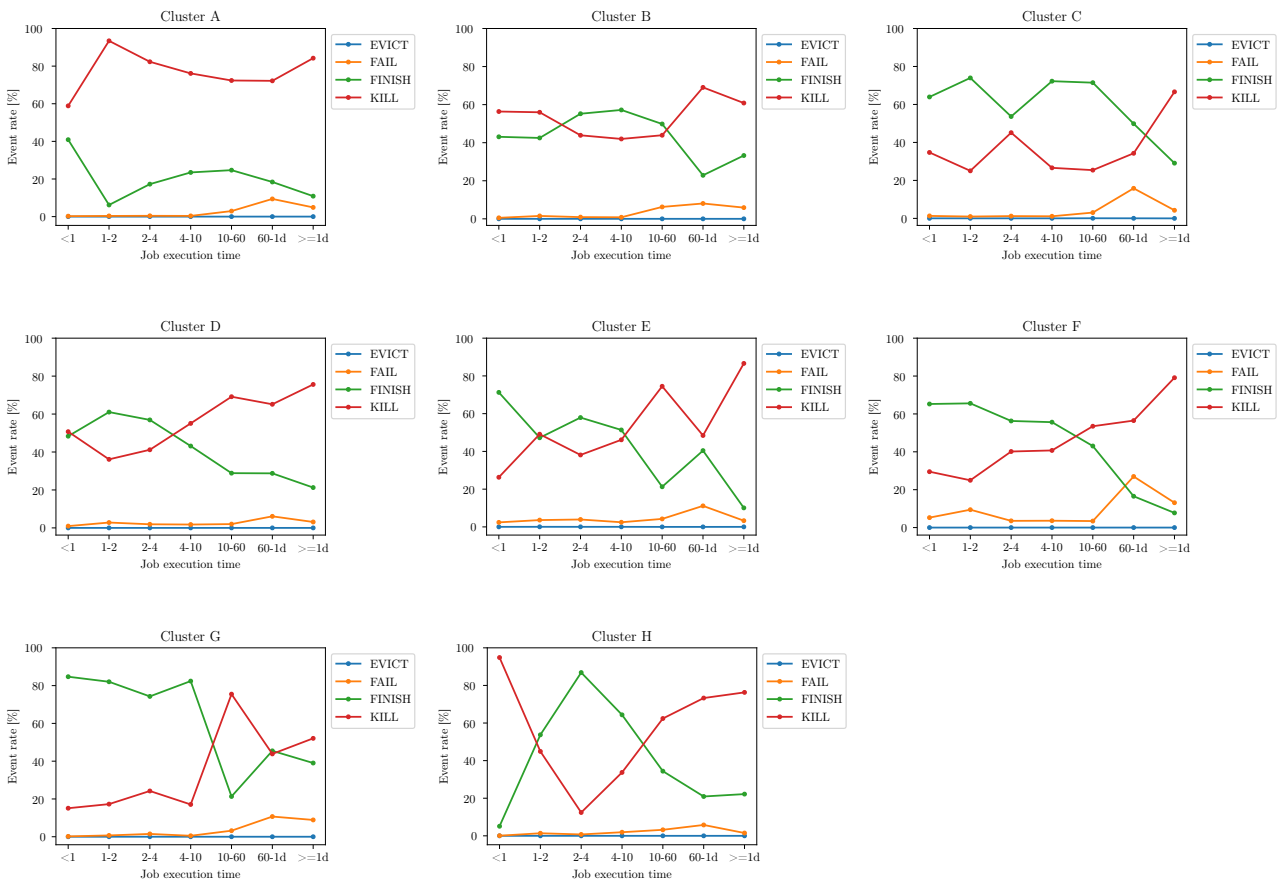
Figure 39. Job event rates vs. job size and final job termination in 2011 and 2019 (all clusters aggregated) traces. The job size is defined as the number of tasks belonging to the job.
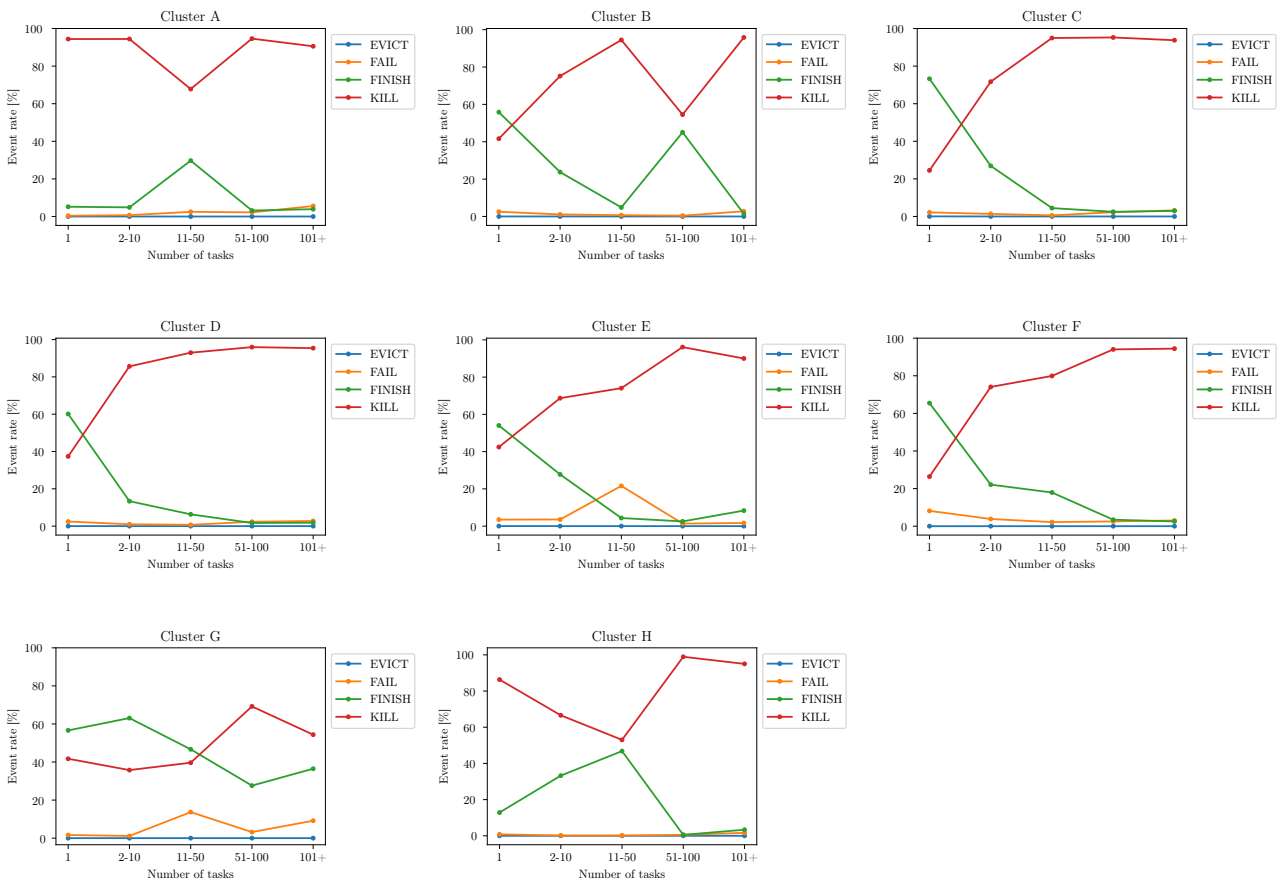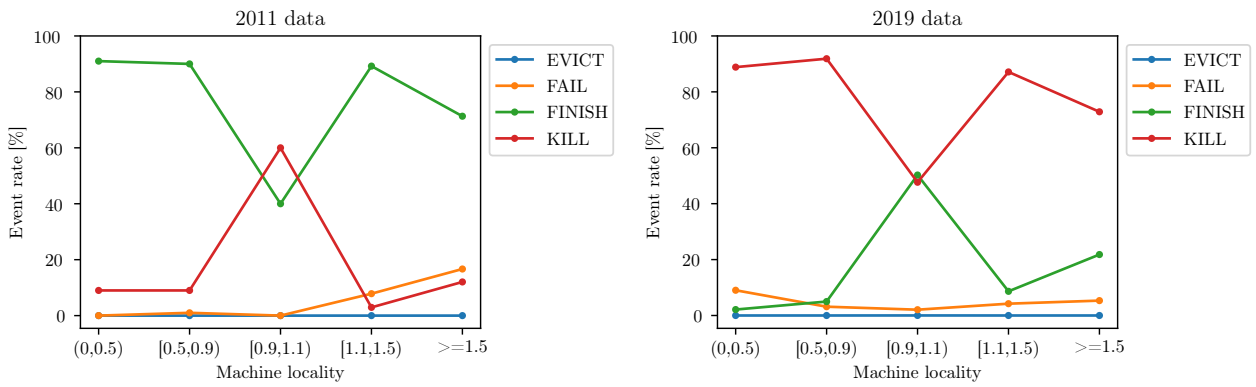

Figure 40. Job event rates vs. job size and final job termination for each 2019 cluster. Refer to figure 39 for the definition of "job size".
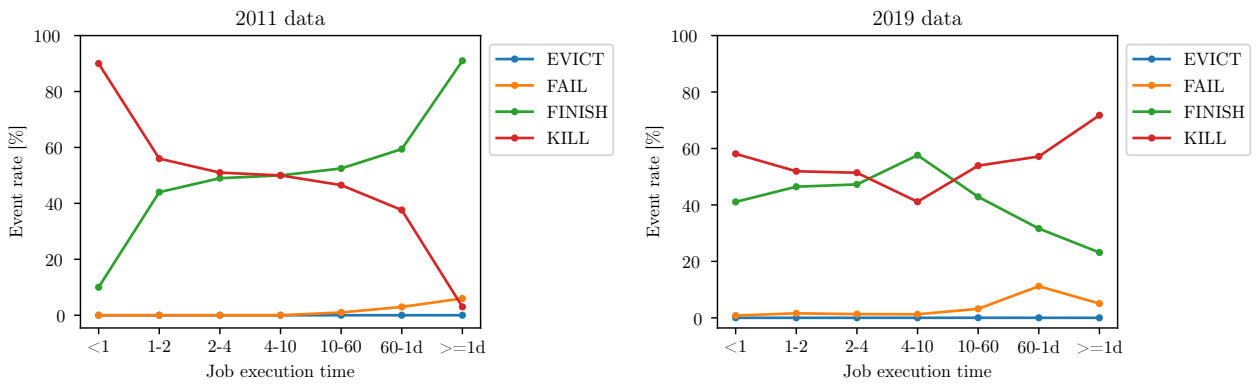
Figure 41. Job event rates vs. event execution time and final job termination in 2011 and 2019 (all clusters aggregated) traces. Execution time classes are defined in minutes, with the exception of "1d" which means one day.

Figure 42. Job event rates vs. event execution time and final job termination for each cluster in the 2019 traces. Refer to figure 41 for interpretation of the execution time classes

Figure 43. 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).
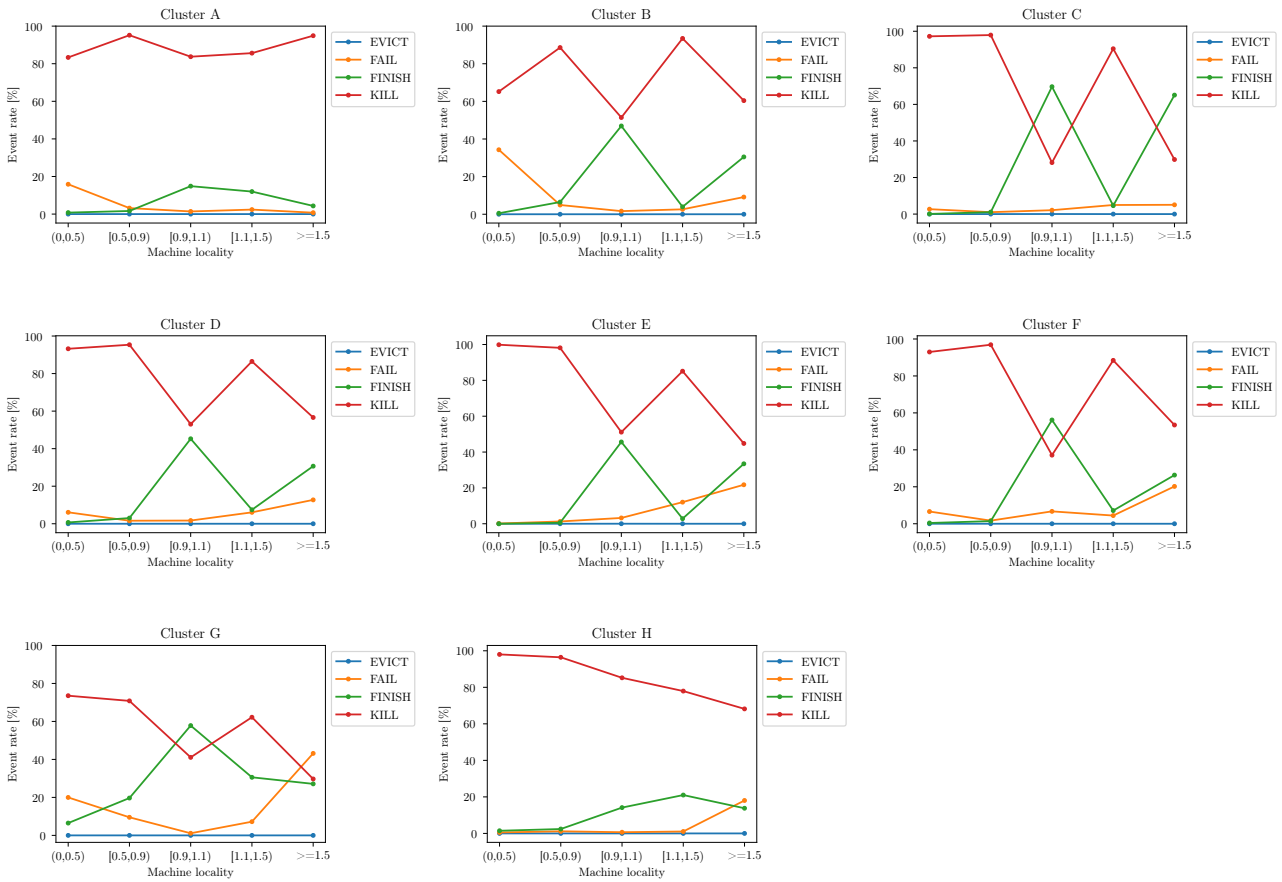
Figure 44. Job event rates vs. machine locality and final job termination for each cluster in the 2019 traces. Refer to figure 43 for the definition of "machine locality".

8 Conclusions, Limitations and Future Work

In this report we analyzed the Google Borg 2019 traces and compared them with their 2011 counterpart from the perspective of unsuccessful executions, their impact on resources and their causes. We discover that the impact of unsuccessful executions (especially of KILLed tasks and jobs) in the new traces is still very relevant in terms of machine time and resources, even more so than in 2011. We also discover that unsuccessful job and task event patterns still play a major role in the overall execution success of Borg jobs and tasks. We finally discover that unsuccessful job and task event rates dominate the overall landscape of Borg's own logs, even when grouping tasks and jobs by parameters such as priority, resource request, reservation and utilization, and machine locality.

We then can conclude that the performed analysis show many clear trends regarding the correlation of execution success with several parameters and metadata. These trends can potentially be exploited to build better scheduling algorithms and new predictive models that could understand if an execution has high probability of failure based on its own properties and metadata. The creation of such models could allow for computational resources to be saved and used to either increase the throughput of higher priority workloads or to allow for a larger workload altogether.

The biggest limitation and threat to validity posed to this project is the relative lack of information provided by Google on the true meaning of unsuccessful terminations. Indeed, given the "black box" nature of the traces and the rather scarcity of information in the traces documentation [4], it is not clear if unsuccessful executions yield any useful computation result or not. Our assumption in this report is that unsuccesful jobs and tasks do not produce any result and are therefore just burdens on machine time and resources, but should this assumption be incorrect then the interpretation of the analyses might change.

Given the significant computational time invested in obtaining the results shown in this report and due to time and resource limitations, some of the analysis were not completed on all clusters. Our future work will focus on finishing these analysis, computing results for the missing clusters and obtaining an overall picture of the 2019 Google Borg cluster traces w.r.t. failures and their causes.

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] John Wilkies. *Borg cluster traces from Google*. Aug. 2020. URL: `https://github.com/google/cluster-data`.

[6] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, et al. "Autopilot: Workload Autoscaling at Google". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys '20. Heraklion, Greece: Association for Computing Machinery, 2020. ISBN: 9781450368827. DOI: `10.1145/3342195.3387524`. URL: `https://doi.org/10.1145/3342195.3387524`.

[7] Dimuth Lasantha and Biplob Ray. "Priority Based Modeling and Comparative Study of Google Cloud Resources between 2011 and 2019". In: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2020, pp. 1310–1317. DOI: `10.1109/TrustCom50675.2020.00176`.

[8] Yuhui Lin, Adam Barker, and Sheriffo Ceesay. "Exploring Characteristics of Inter-cluster Machines and Cloud Applications on Google Clusters". In: *2020 IEEE International Conference on Big Data (Big Data)*. 2020, pp. 2785–2794. DOI: `10.1109/BigData50022.2020.9377802`.

[9] Bianca Schroeder and Garth A. Gibson. "A Large-Scale Study of Failures in High-Performance Computing Systems". In: *IEEE Transactions on Dependable and Secure Computing* 7.4 (2010), pp. 337–350. DOI: `10.1109/TDSC.2009.4`.

[10] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. "DRAM Errors in the Wild: A Large-Scale Field Study". In: *Commun. ACM* 54.2 (Feb. 2011), pp. 100–107. ISSN: 0001-0782. DOI: `10.1145/1897816.1897844`. URL: `https://doi.org/10.1145/1897816.1897844`.

[11] Shan Lu, Soyeon Park, Eunsoo Seo, et al. "Learning from Mistakes: A Comprehensive Study on Real World Concurrency Bug Characteristics". In: *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XIII. Seattle, WA, USA: Association for Computing Machinery, 2008, pp. 329–339. ISBN: 9781595939586. DOI: `10.1145/1346281.1346323`. URL: `https://doi.org/10.1145/1346281.1346323`.

[12] Ding Yuan, Soyeon Park, Peng Huang, et al. "Be Conservative: Enhancing Failure Diagnosis with Proactive Logging". In: *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*. OSDI'12. Hollywood, CA, USA: USENIX Association, 2012, pp. 293–306. ISBN: 9781931971966.

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