Borg: the Next Generation

Muhammad Tirmazi
Harvard University
tirmazi@g.harvard.edu

Adam Barker[*]
University of St Andrews
adb20@st-andrews.ac.uk

Nan Deng
Google
dengnan@google.com

Md E. Haque
Google
mehaque@google.com

Zhijing Gene Qin
Google
geneqin@google.com

Steven Hand
Google
sthand@google.com

Mor Harchol-Balter
CMU
harchol@cs.cmu.edu

John Wilkes
Google
johnwilkes@google.com

Abstract

This paper analyzes a newly-published trace that covers 8 different Borg [35] clusters for the month of May 2019. The trace enables researchers to explore how scheduling works in large-scale production compute clusters. We highlight how Borg has evolved and perform a longitudinal comparison of the newly-published 2019 trace against the 2011 trace, which has been highly cited within the research community.

Our findings show that Borg features such as alloc sets are used for resource-heavy workloads; automatic vertical scaling is effective; job-dependencies account for much of the high failure rates reported by prior studies; the workload arrival rate has increased, as has the use of resource over-commitment; the workload mix has changed, jobs have migrated from the free tier into the best-effort batch tier; the workload exhibits an extremely heavy-tailed distribution where the top 1% of jobs consume over 99% of resources; and there is a great deal of variation between different clusters.

Keywords. Data centers, cloud computing.

ACM Reference Format:
Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: the Next Generation. In *Fifteenth European Conference on Computer Systems (EuroSys '20), April 27–30, 2020, Heraklion, Greece.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3342195.3387517

1 Introduction

Data centers are huge capital investments that serve a wide range of applications including search engines, video processing, machine learning, and third-party cloud applications. Modern cluster management systems [22, 34, 35] have evolved to manage these data centers efficiently, and several companies have published job-scheduling *traces* of their cluster management systems so that external researchers can explore how they achieve this. Examples include Google [36], Microsoft [3] and Alibaba [33].

Date	*May 2011*	*May 2019*
Duration	30 days	31 days
Cells (clusters)	1	**8**
Machines	12.6k	**96.4k**
Machines per cell	12.6k	12.0k
Hardware platforms	3	7
Machine shapes	10	**21**
Priority values	0–11	**0–450**
Alloc sets	–	**Y**
Job dependencies	–	**Y**
Batch queueing	–	**Y**
Vertical scaling	–	**Y**
Format	csv files	BigQuery tables

Table 1. Comparison between 2011 and 2019 traces.

In 2011, Google published a 1-month trace from its Borg cluster management system to enable external researchers to explore how scheduling worked in a large-scale compute cluster. Several hundred researchers have taken advantage of that trace to study a wide range of phenomena. An open question is how workloads for such systems evolve over time, and how the evolution of the cluster managers has affected scheduling decisions. To support research on these topics, Google has published a new "2019" trace [37] that includes detailed Borg job scheduling information from 8 different Google compute clusters (Borg cells) for the entire month of May 2019. The new trace contents are an extension of the data provided in the 2011 trace [28, 36]. This paper reports on the results of a first longitudinal comparison between the 2011 and 2019 traces, repeating several analyses from the first paper to analyze the 2011 trace [27], and adding some new ones. We also discuss new features in Borg that were not visible in the 2011 trace. Our key observations can be summarized as follows:

1. **The 2019 trace has several new features (§3 and §5)**, including information about allocations [35], parent-child dependencies (which affects how task exits are to

[*] Work done whilst on sabbatical leave as a Visiting Researcher at Google.

EuroSys '20, April 27–30, 2020, Heraklion, Greece
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6882-7/20/04.
https://doi.org/10.1145/3342195.3387517

be interpreted), and additional placement constraints. It also covers eight different cells, not just one.

2. **The workload mix has changed (§4)**: much of the workload has moved from the free tier (low priority) into the best-effort batch tier (jobs managed by a queued batch scheduler), while the overall usage for production tier (high priority) jobs has remained approximately constant. We also observe considerable inter-cell workload variation.
3. **The scheduling rate is higher (§6)**: the job submission rate is 3.7× higher than in 2011, and the number of tasks needing to be scheduled has increased by a factor of 7, even though the sizes of the 2019 cells are comparable to the 2011 cell. Despite this, the time the scheduler takes to find places to host the tasks is largely unchanged.
4. **A very heavy-tailed distribution of job sizes (§7):** the compute and memory consumption of jobs are *extremely* variable, with squared coefficients of variation over 23000. Both compute and memory consumption follow power-law Pareto distributions with very heavy tails: the top 1% of jobs ("resource hogs") consume over 99% of all resources, which has major implications on scheduler design: care is needed to insulate the bottom 99% of jobs (the "mice") from the hogs to keep queueing times under control.
5. **Vertical autoscaling is effective (§8):** the new traces demonstrate how automated vertical scaling of per-task resources provides noticeable savings. (A companion paper [29] explores this in much greater depth, with a focus on automated memory scaling.)

We begin with a quick overview of Borg, followed by information about the new 2019 trace, highlighting changes from the 2011 trace.

2 A quick summary of Borg

(This description is a highly abbreviated summary of the information in [35].) Like many similar systems [13, 22, 34], a Borg deployment comprises a logically centralized cluster scheduler *master*, and a large number of machines (or *nodes*), each of which runs a local management agent. Each such deployment is called a *cell*, and is operated as a single management unit. Both the 2011 and the 2019 traces consist of time-stamped data from the master and the individual machines.

The cluster scheduler receives job creation requests submitted on behalf of a *user* (an internal developer or a group, used for accounting and authentication purposes); these requests identify the executable binaries along with additional details such as the resources needed, the number of replicas, the job priority, and so on. In Borg, each replica in a job is called a *task*, and the role of the cluster scheduler is to place each task onto a suitable machine (i.e., one where the requisite resources are available). Tasks inherit several properties from their job: for example, all tasks within a job have the same priority, and if a job is queued then all of its constituent tasks are also put into a queued state.

The cluster scheduler will assign an upper bound on the resource consumption of a task on a machine. This upper bound is called the *limit* and is configured based on the requested resources; the actual resource *usage* of a task varies over time. For memory, the limit is generally a hard bound (i.e., the usage will never exceed the limits), but for CPU the system is typically configured to be work conserving, meaning the CPU usage can exceed the assigned limits if the machine's CPU is not overloaded. In theory, the sum of limits of all running tasks should never exceed the machine's capacity.

The priority of a job helps define how the scheduler treats it. Ranges of priorities that share similar properties are referred to as *tiers*:

- **Free tier**: jobs running at these lowest priorities incur no internal charges, and have no Service Level Objectives (SLOs). 2019 trace priority <= 99; 2011 trace priority bands 0 and 1.
- **Best-effort Batch (beb) tier**: jobs running at these priorities are managed by the batch scheduler and incur low internal charges; they have no associated SLOs. 2019 trace priority 110–115; 2011 trace priority bands 2–8.
- **Mid-tier**: jobs in this category offer SLOs weaker than those offered to production tier workloads, as well as lower internal charges. 2019 trace priority 116–119; not present in the 2011 trace.
- **Production tier**: jobs in this category require high availability (e.g., user-facing service jobs, or daemon jobs providing storage and networking primitives); internally charged for at "full price". Borg will *evict* lower-tier jobs in order to ensure production tier jobs receive their expected level of service. 2019 trace priority 120–359; 2011 trace priority bands 9–10.
- **Monitoring tier**: jobs we deem critical to our infrastructure, including ones that monitor other jobs for problems. 2019 trace priority >= 360; 2011 trace priority band 11. (We merged the small number of monitoring jobs into the Production tier for this paper.)

3 Comparing the 2011 and 2019 traces

This section compares the 2011 and 2019 traces at a high level; there is considerably more information in the new trace documentation [37].

Table 1 provides a summary of the main changes. The older trace described a single cell, while the newer trace spans eight. The average number of machines per cell remains similar, although there is greater variety in the CPU-to-memory ratio, as illustrated in Figure 1.

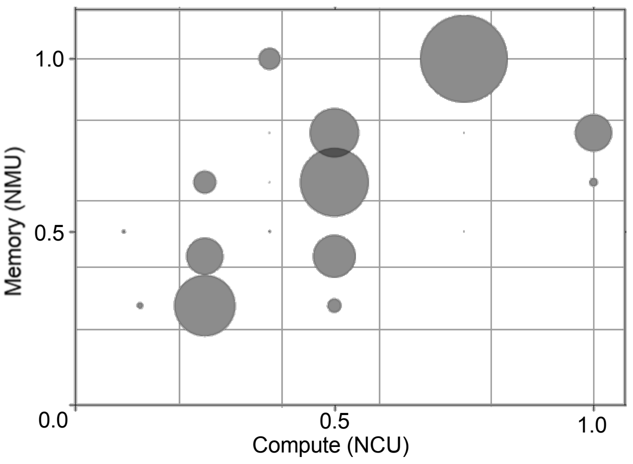


Figure 1. Frequency of machine shapes as a function of CPU and memory (RAM) capacity. The area of each circle is proportional to the number of machines. Both CPU and memory are re-scaled to a 0–1 range in the trace.

Job priority values: the new trace exposes the raw priorities as sparse values in the range 0–450; the 2011 trace mapped the unique job priority values (0, 25, 100, 101, 103, 104, 107, 109, 119, 200, 360, 450) to the integers from 0–11; i.e., the value 3 in the 2011 trace corresponds to a raw priority of 101. (The meaning of some of the priority values has changed since 2011. We follow the trace documentation [37] in mapping jobs to the tiers listed above.)

CPU usage histograms: the 2019 trace adds a 21-element histogram of CPU utilization for each 5 minute sampling period, biased towards high percentiles.

Normalized CPU units: to help its users better handle machine heterogeneity, Borg has switched to using abstract *Google Compute Units (GCUs)* instead of physical CPU cores, and the trace follows suite. An allocation of 1 GCU should provide about the same amount of computational power on any machine in the fleet, and Borg maps that onto the appropriate number of physical CPU cores. As in 2011, the 2019 trace further re-scales these values based on the maximum machine sizes in the trace, so that the resulting *Normalized Compute Units (NCUs)* are always in the range 0–1.

Alloc sets: these allow users to reserve resources on machines into which jobs can later be scheduled – e.g., to secure resources before bringing up a new workload, to retain resources while jobs are turned down and back up again, and to co-locate tasks from different jobs [35]. The 2011 trace elided data about alloc sets, and treated them (and the jobs that landed inside them) as if they were jobs. The new trace provides information about both jobs and alloc sets (together called *collections*) and how tasks are mapped into alloc instances (*instances*).

Job dependencies: if a Borg job has a *parent job*, the child job will be killed automatically when its parent terminates. This simplifies job cleanup for systems like MapReduce [10] where a controller job spawns a number of child workers that are automatically removed when the controller exits. The 2019 traces provide this dependency data, permitting more sophisticated failure analyses.

Batch queuing: like Omega [31], Borg now supports multiple schedulers, including a *batch scheduler* that manages the aggregate batch job workload for throughput by *queueing* jobs until the cell can handle them, after which the job is given to the regular Borg scheduler.

Vertical scaling: Borg now supports vertical autoscaling via a system called Autopilot that automatically predicts a job's resource usage, and adjusts its resource limits dynamically [29]. The trace indicates which jobs were subject to this autoscaling.

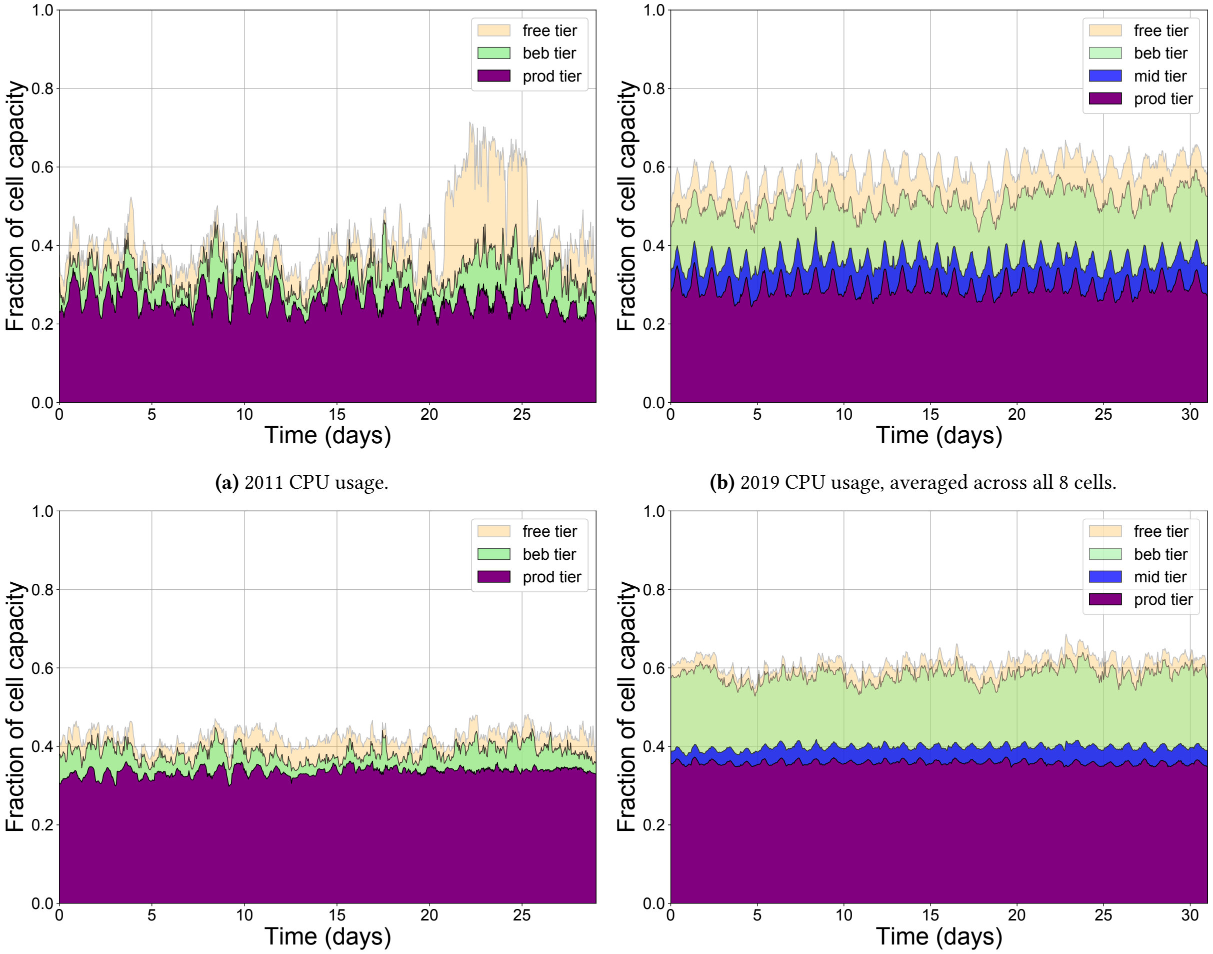
The 2019 trace contains an average of 350 GiB of compressed data per cell, compared to about 40 GiB for the 2011 trace, or 2.8 TiB of data for the entire trace. To obviate the need to download so much data, and to simplify analyses, the new one is provided as data tables in Google BigQuery [5], which is an externally-available version of Google's Dremel system [24] that provides both storage and analytic tools built around a dialect of SQL. Most of our analyses were done using BigQuery, and we also imported the 2011 trace to BigQuery, so we could use the same queries in both cases. This allowed us to validate many of our evaluations, including reproducing the data reported in [27].

Our results are presented both as simple time series aggregations as well as *Complementary Cumulative Distribution Functions* (CCDFs [6]) to visually summarize distributions of data. These show the fraction of samples that have a value larger than the one at a given point on the x-axis.

Both the 2011 and 2019 traces normalized resource units for CPU and memory so that they scale from 0–1. The absolute machine sizes are different in the two traces, so the values cannot directly be compared, but relative comparisons are still valid, such as percentage utilizations.

4 Resource utilization

Figure 2 shows the average compute and memory *usage* for every hour of the 2011 and 2019 traces. The average utilization has increased over 8 years, mostly due to an increase in consumption from the best-effort batch tier across both resource dimensions. Best-effort batch now accounts for ∼ 20% of the cell's capacity for both CPU and memory, which is a considerable increase over the 2011 trace, although somewhat countered by a reduction in the usage from the free tier. (The lower variance visible in the 2019 trace is partly a result of aggregation across multiple cells.)



(a) 2011 CPU usage.

(b) 2019 CPU usage, averaged across all 8 cells.

(c) 2011 memory usage.

(d) 2019 memory usage, averaged across all 8 cells.

Figure 2. The fraction of a cell's total resource capacity **used** during each hour-long interval.

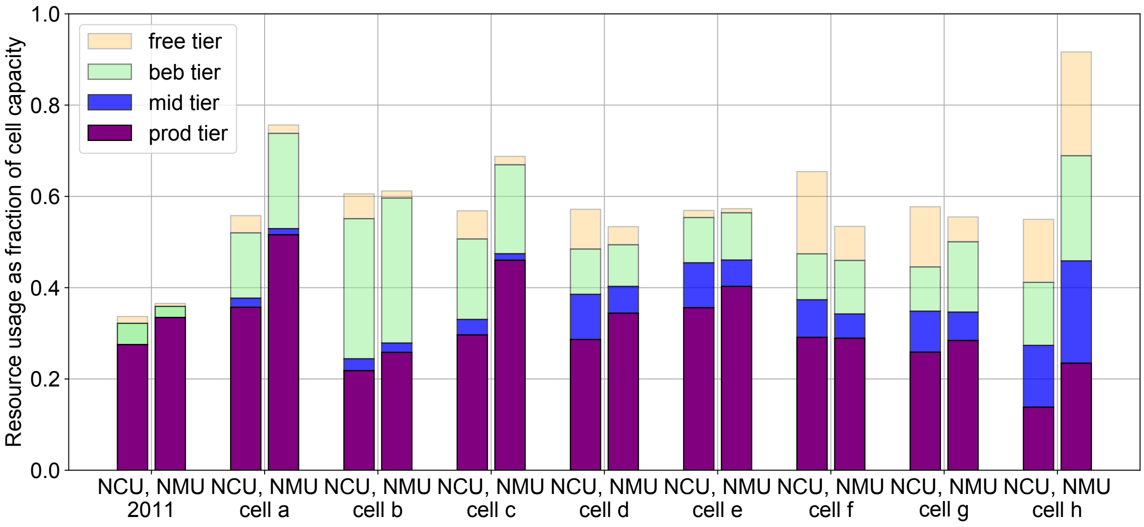
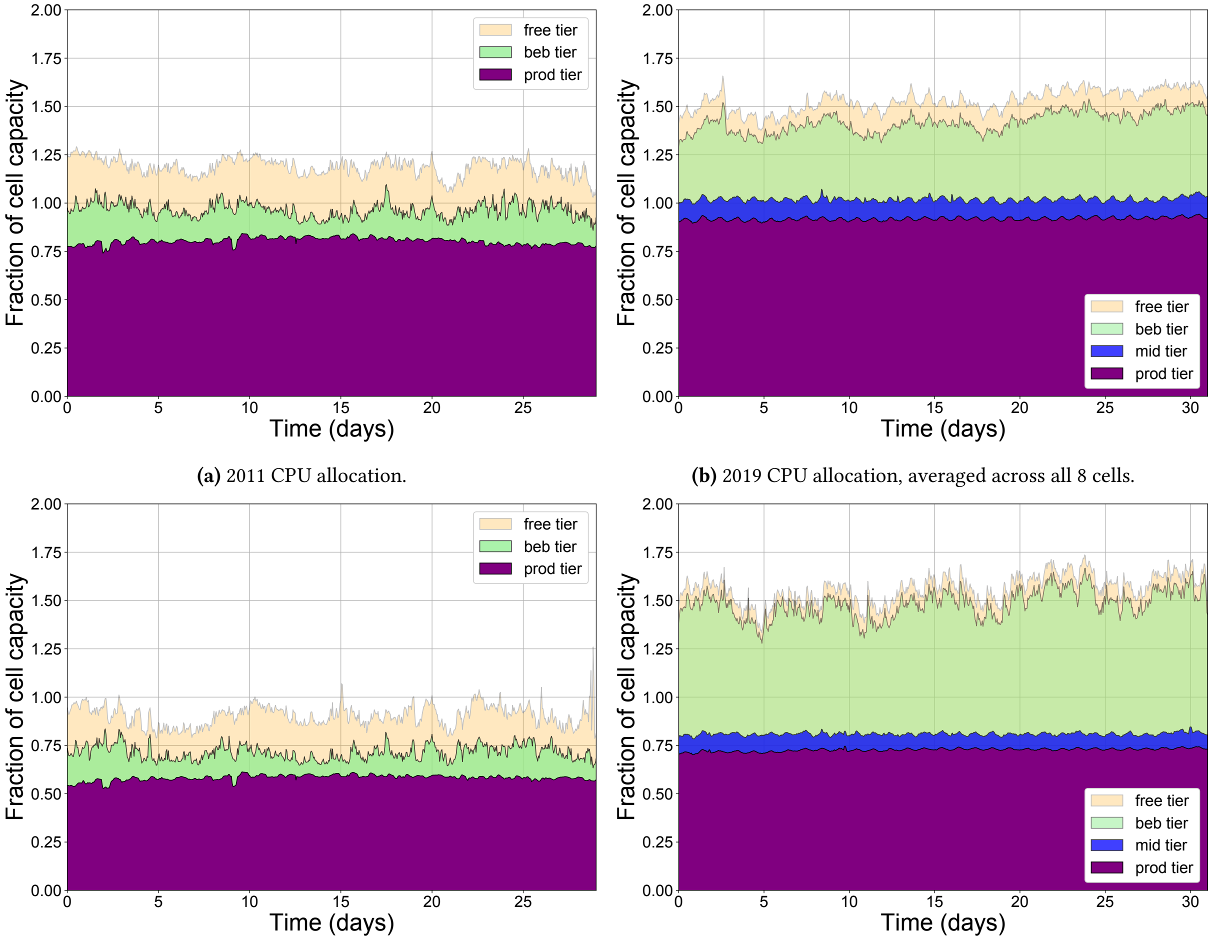


Figure 3. Average **utilization** (as a fraction of cell capacity across the entire trace duration) by tier, for 2011 and the 8 cells in the 2019 trace.



(a) 2011 CPU allocation.

(b) 2019 CPU allocation, averaged across all 8 cells.

(c) 2011 memory allocation.

(d) 2019 memory allocation, averaged across all 8 cells.

Figure 4. The fraction of a cell's total resource capacity **allocated** during each hour-long interval.

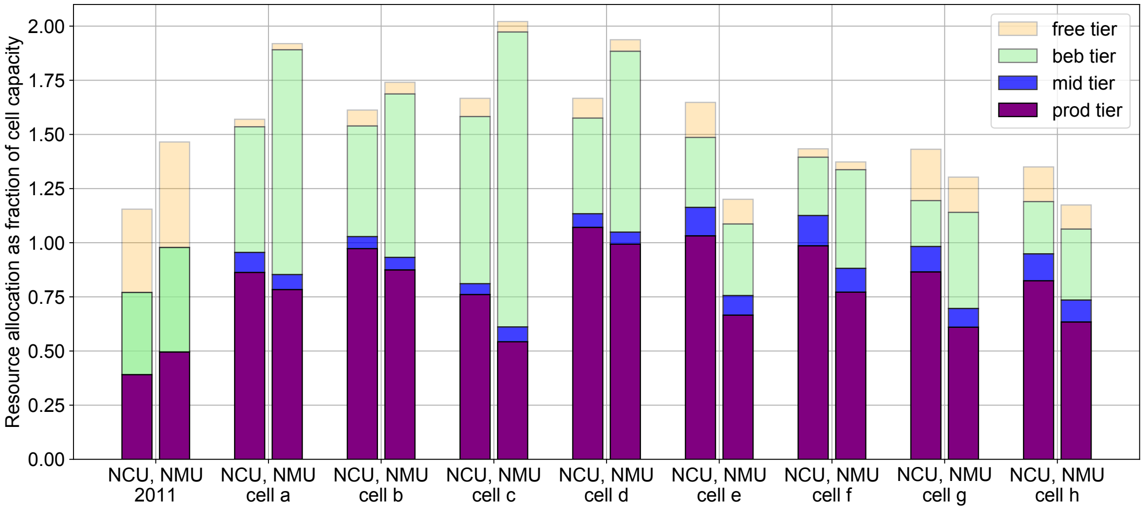


Figure 5. Average **allocation** (as a fraction of cell capacity across the entire trace duration) by tier, for 2011 and the 8 cells in the 2019 trace.

Figure 3 shows how the CPU and memory usage vary across the eight cells, broken up by tier; the two bars on the far left of the graph show data from the 2011 trace. As expected [35], there is considerable variation in the workload mix from cell to cell, for example, cell b has the largest proportion of best-effort batch jobs, cell a has the largest proportion of production jobs and cell h has the largest proportion of mid-tier jobs. There is also variation *within* a cell across the different resource dimensions, for example cells a and h both exhibit large variation between memory and CPU resource usage.

Jobs also specify a requested limit (upper bound) on the amount of resources they will need, and this quantity is used to determine whether or not an instance can fit on a particular machine. Figure 4 shows that the sum of these limits has significantly increased from 2011 to 2019, with both resource dimensions now being consistently allocated well above 100% of the deployed capacity. This demonstrates that Borg is using statistical multiplexing to over-commit resources, i.e., betting that jobs will under-use the resources they request. In 2011, CPU was more aggressively over-committed than memory, because the penalty for short-term CPU over-allocation is throttling, while insufficient RAM on a machine will cause an out of memory (OOM) eviction. That is no longer true in 2019: the amount of memory over-allocation is comparable to that for CPU.

At this level of aggregation (1-hour averages across tens of thousands of machines), Borg has not significantly increased the fraction of machine resources that are actually consumed since 2011, but it has successfully increased the amount of work that its internal users can place on a fixed amount of machine capacity. This means that less additional compute capacity needed to be purchased to cope with a growing potential workload.

The tiers exhibit different behaviors: for example, compute usage at the production tier is only around 30% of the allocated level, while the memory usage is closer to 65%; for the mid-tier, the allocation and usages are closer together.

Figure 5 shows how the CPU and memory allocations vary across the eight cells, broken up by tier; the two bars on the far left show data from the 2011 trace. We see considerable inter-cell variation and some very highly over-allocated cells, for example, cell c has allocated $\sim$ 140% of the cell's memory capacity just to the best-effort batch tier.

4.1 Machine utilization

Figures 6a and 6b show CCDFs of machine CPU and memory utilization for all 8 cells at the same local time in the 2019 trace for each cell, along with data from the 2011 trace.

The first thing to note is that the machine utilization in 2019 is higher when compared to 2011 at all but the highest percentiles: the median overall utilization has increased by 20–40% for CPU and 3–30% for memory. Secondly, the variation in utilization is lower in the 2019 cells compared with 2011 (visible as steeper lines in many of the CCDF lines). Thirdly, there are fewer machines with CPU utilization >80% than in 2011. Put together, this means that the Borg scheduler is doing a better job in 2019 than it did in 2011 in terms of distributing workload across the cell and avoiding both under- and over-utilizing machines.

The next observation is that there is considerable variation across the 2019 cells for both memory and CPU at all utilization levels (e.g., almost 30% utilization at the median). This aligns with the importance of heterogeneity that was described in the original Borg paper [35].

We also observed a diurnal cycle in the loads, corresponding to Figure 2: the load at midnight PDT was much higher in cell *g* in Singapore where it was 3pm, than in the others where it was 2 or 3am locally in the USA.

5 New Borg features and trace properties

This section discusses some of the changes in Borg's behavior between 2011 and 2019 that are reflected in the traces.

5.1 Alloc sets

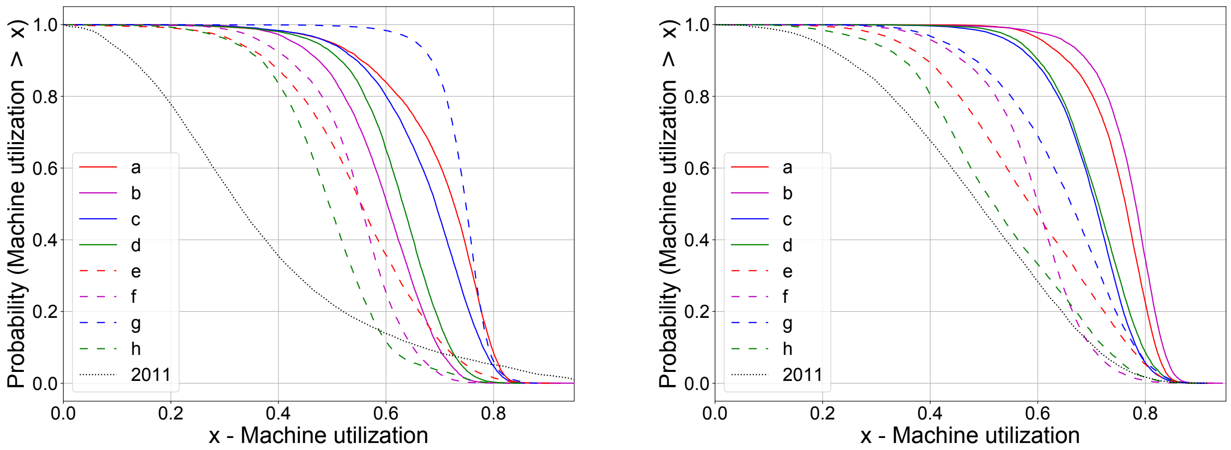
An alloc set is a set of reserved resources (alloc instances) into which jobs can be scheduled, by placing a job's tasks inside the alloc instances [35]. Because the 2011 trace was made available before [35] was published, alloc information was not included in that trace, but is included in the 2019 one.

A *collection* is an alloc set or a job. Only 2% of collections are alloc sets, but they are an important feature of production workloads: alloc sets represent 20% of the total CPU allocations and 18% of the RAM. 15% of the jobs are marked to run in an alloc set, most of which (95%) are from the production tier. Jobs within allocs have a higher average memory utilization (73%) than other jobs (41%).

5.2 Terminations

Many authors (e.g., [2, 11, 16, 32]) have reported surprisingly high task failure rates in the 2011 trace. For example: "Unsuccessful job terminations in the Google trace are 1.4–6.8× higher than in other traces" [11]. We suspect that this may have been an unfortunate side-effect of omitting some crucial information in the 2011 trace and its documentation: the majority of "failures" are transitions triggered directly or indirectly by users canceling jobs. A collection or an instance can be terminated through one of the following four events:

- FINISH: the collection or instance completed normally (aka "success").
- EVICT: the collection or instance was de-scheduled due to a (rare) hardware failure, or a forced OS upgrade (about 1/month per machine), or preempted by a higher priority instance, or because the machine was over-committed and Borg had to kill one or more instances to free enough resources for the rest. Most



(a) CPU utilization **(b)** Memory utilization

Figure 6. Machine CPU and memory utilization (usage ÷ size) for all machines on the 15th day of the trace at the same time of day (1:00 to 1:05pm in the local timezone, apart from cell g, for which we used 12 noon to 12:05pm because it is in Singapore, which does not observe daylight savings time). The letters correspond to different cells in the 2019 trace.

evictions are of instances, not collections. In almost all cases, an evicted instance will be rescheduled elsewhere in the same cell.

- KILL: the collection or instance was canceled by the user, either directly via an RPC to the Borgmaster, or indirectly if this was a child of a parent job that exited or was killed.
- FAIL: the collection or instance was unexpectedly terminated because it had a problem of its own, such as a segfault, or trying to use more resources than it had requested (e.g., from a memory leak or misconfiguration).

Of all these event types, only EVICTs are caused by the cluster infrastructure. And even in these circumstances, Borg has maximum-permitted eviction rate SLOs to protect important collections from being evicted too often. This is supported by the trace data: of the 24.8M collections in the 2019 trace, only 0.79M of them (3.2%) experience any instance evictions, and 96.6% of those collections are in non-production tiers (priority <120). For jobs in the production tier, <0.2% of collections have any instances evicted, and 52% of these collections experience only a single eviction.

Users initiate most KILL events, and the traces do not have an explanation of why this was done. One exception is job dependencies, which is a commonly used mechanism among Borg users: across all 8 cells, we have observed a higher percentage (87%) of jobs with parent experiencing a KILL event compared to jobs without any parent (41%). Providing parent information in the 2019 trace partly explains the high KILL rates reported earlier.

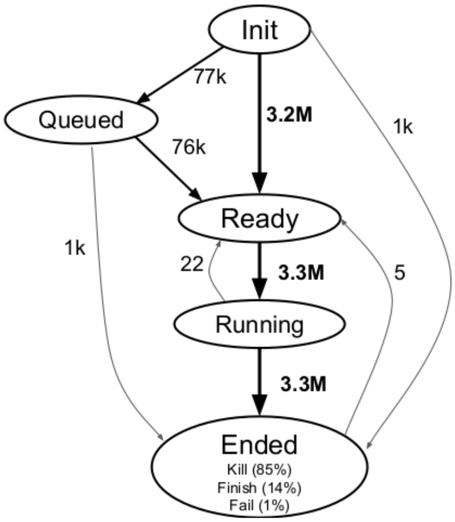


Figure 7. The job state transitions observed between jobs in cell g, with occurrence counts for the transitions. The "unusual" transitions are extremely rare compared to the common ones.

5.3 State transitions

Figure 7 shows the state transition diagram for collections and instances, annotated with the relative frequencies of the transitions. As expected, common paths are many orders of magnitude more frequently exercised than the rarer ones.

6 Evolution in the scheduling load

This section reports some analyses of how the scheduler load has evolved since 2011.

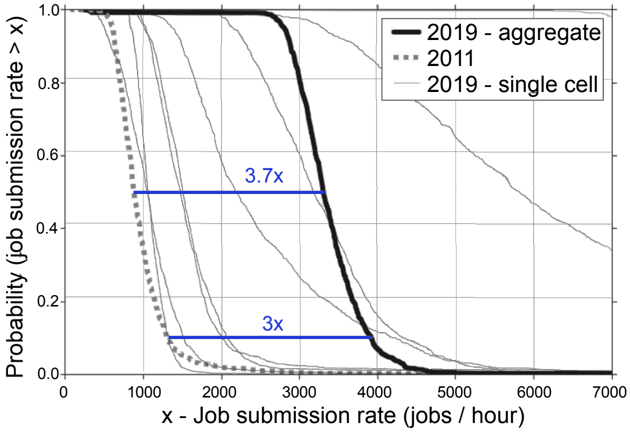


Figure 8. CCDF of the mean job submission rate per hour to the Borg scheduler per cell. The 2019 data is shown both as a rate averaged across all cells, and also on a per-cell basis.

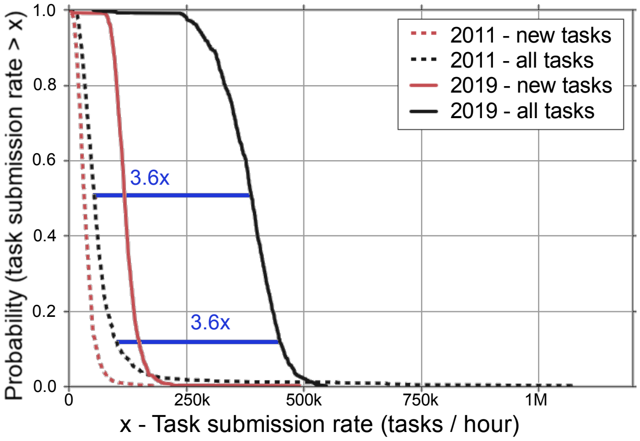


Figure 9. CCDF of the number of tasks/hour submitted to the Borg scheduler. *New tasks* are ones that are members of newly-submitted jobs; *all tasks* also includes rescheduled tasks that were previously running.

6.1 Job submission rate

Figure 8 shows the CCDF of the number of jobs submitted per hour to the Borg scheduler per cell. Even though the cell sizes are comparable (see Table 1), the mean job arrival rate increased 3.5× from 964 jobs per hour in 2011 to 3360 jobs per hour in 2019; the median arrival rate increased 3.7× from 885 to 3309 jobs per hour; and the 90%ile has grown by approximately 3×.

6.2 Task submission rate

Figure 9 shows the rate at which the Borg scheduler is processing task scheduling decisions, for both *new tasks* and *all tasks*, the latter of which includes tasks that have previously been running but are now being *rescheduled*. The first thing to notice is that the median task scheduling rate has increased drastically since 2011 (by about 3.6× for all tasks); the second is that a lot of the task scheduling events are for rescheduling: the ratio of the median resubmitted task rate to the median new task rate increased from 0.66:1 to 2.26:1. This indicates there is considerably more "churn" in the modern system.

6.3 Scheduling delay

Considering this increase in job and task submission rate, it might be expected that the Borg scheduler could take longer to make scheduling decisions, and/or tasks or jobs could remain stalled, waiting for scheduling decisions. To assess this, we examined the time it took Borg to schedule the first task of a READY job onto a machine (state RUNNING), to exclude deliberate queueing delays due to the batch scheduler. We picked this measure because users do not expect all their tasks to be available before a job can start doing useful work [35]. Figure 10 shows the results: median scheduling delays have actually *decreased*, although the tail is longer for the last 28% of jobs. Most of the long delays are associated with best-effort batch and mid-tier jobs: production jobs are scheduled significantly faster than in 2011. (Also: note that some jobs may experience a delay in the QUEUED state before entering the READY state.)

To understand why this might be happening, we looked at the number of tasks per job by tier (Figure 11): best-effort batch and mid-tier jobs have a larger number of tasks than the other tiers. For example, the 80%ile best-effort batch job has 25 tasks, but other tiers only have 1 task; and at the 95%ile, the values are 498, 67, 21, and 3 tasks for best-effort batch, mid-tier, free, and production tier, respectively. Best-effort batch and mid-tier jobs may take longer to schedule simply because they contain more tasks.

7 Resource consumption

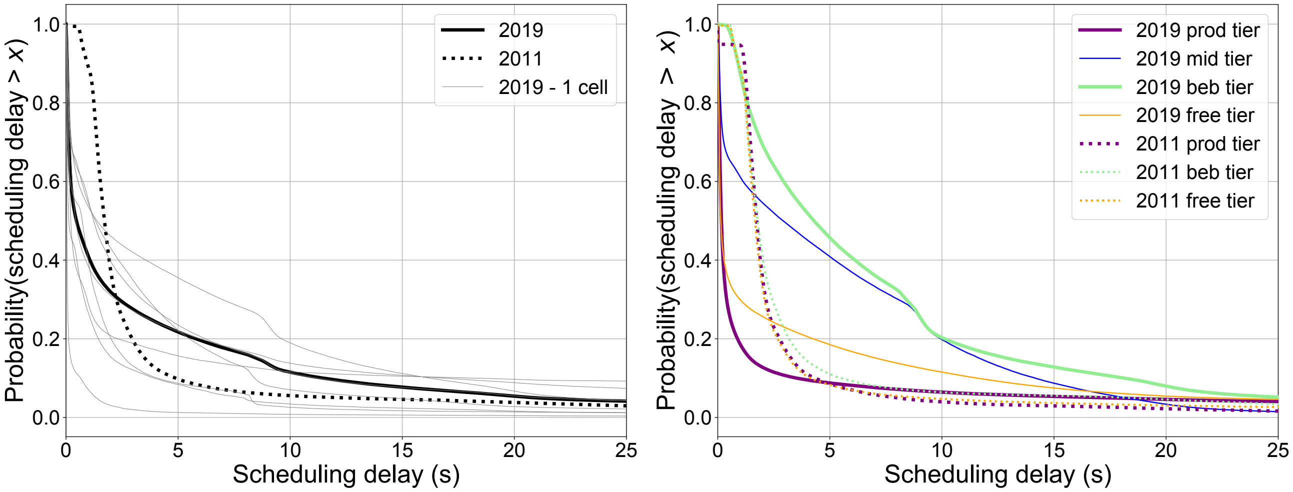
In this section we examine the integral of resource consumption across time for jobs in the 2019 trace. For compute usage, we add up the number of NCU-hours (note: not core-hours) consumed by a job across its lifetime. A 20 NCU-hour job might have consumed 20 machines for an hour, or one machine for 20 hours, or used 20% of 100 machines for an hour, etc. Similarly, for memory usage we total the Normalized Memory Unit-hours (NMU-hours) consumed by a job.

As Table 2 and Figure 12 show, there is huge variability in the number of NCU-hours and NMU-hours used per job. One way to understand this variability is to look at the squared coefficient of variation (C^2), where:

$$C^2 = \text{variance}/\text{mean}^2$$

C^2 is invariant to data normalization, and thus is particularly helpful here.

To give some context, for an exponential distribution, $C^2 = 1$. In 1996, measurements of compute consumption in UNIX jobs at U.C. Berkeley found $C^2 \approx 50$ [19], which was



(a) Job scheduling delay by cell.

(b) Job scheduling delay by tier.

Figure 10. CCDF of job scheduling delay, i.e., the time from when a job is in the READY state until its first task is running. For 2019, we aggregate the data across all 8 cells, and in (a) also provide the per-cell data.

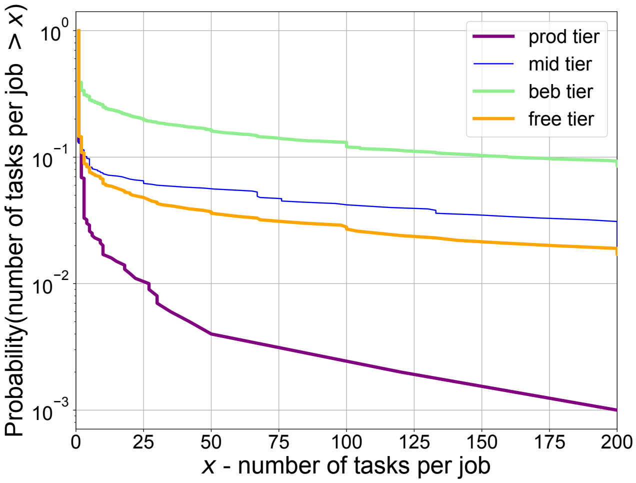


Figure 11. CCDF of the number of tasks per job by tier for the 2019 trace. Note the log-scale y-axis.

	NCU-hours		*NMU-hours*	
Measure	*2011*	*2019*	*2011*	*2019*
median	0.15e-3	0.05e-3	0.07e-3	0.03e-3
mean	3.00	1.19	3.00	0.67
variance	75.2k	33.3k	99.0k	19.8k
90%ile	0.03	0.005	0.01	0.004
99%ile	10.5	1.33	5.2	0.65
99.9%ile	248	69.67	196	36.6
maximum	138 k	370 k	151 k	299 k
top 1% jobs load	97.3%	99.2%	98.6%	99.1%
top 0.1% jobs load	83.0%	93.1%	89.3%	92.6%
C^2	8375	23 312	11 001	43 476
Pareto(α)	0.77	0.69	0.72	0.72
R^2	99.8%	99.9%	99.8%	99.6%

Table 2. Distribution data for the integral of resource usage by job, for CPU (NCU-hours) and memory (NMU-hours). The Pareto distribution data is for large jobs: ones where NCU-hours and NMU-hours were $>$ 1 and $<=$ 99.99%ile value. All percentiles and C^2 values are based on unbiased random samples.

considered to be extremely high at the time. A 2005 study of workloads in several supercomputing centers found that C^2 ranged from 28 to 256, depending on the supercomputing center [23], and another 2004 study of supercomputing centers found $C^2 = 43$ [30]. A 2017 study of the sizes of a billion objects being cached at Akamai touted “extremely high variability” with $C^2 = 143$ for a Hong Kong trace and $C^2 = 760$ for a U.S. trace, with object sizes spanning 9 orders of magnitude [4].

Google’s 2019 workloads in the trace are 1–2 orders of magnitude more variable than even the Akamai measurements above. For compute consumption, our variance is about 33.3k with a mean of only 1.2 NCU-hours, so we end up with $C^2 = 23$k. For memory, the story is even more extreme: the variance is about 19.8k for a mean of 0.67 NMU-hours, leading to $C^2 = 43$k. Both of these C^2 values are incredibly high.

The (relatively) straight lines on the log-log scale in Figure 12 tell us that resource-hour consumption has a power-law distribution; specifically, it is Pareto(α) distributed [18]:

$$\mathbf{Pr}\{\text{job uses} > x \text{ NCU-hours}\} = 1/x^{\alpha}$$

(and similarly for NMU-hours) where α is the negative slope of the line. If we consider only “larger” jobs (ones that use more than 1 NCU-hour for CPU consumption or 1 NMU-hour

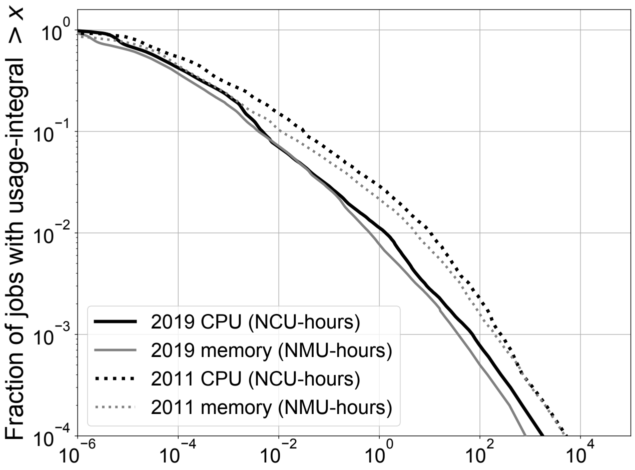


Figure 12. CCDF of resource-usage-hours by job for 2011 and 2019 – i.e., the fraction of jobs that use at least x resource-hours. Note the log-log scale.

for memory), and ignore the outliers at the very end of the tail (the top 0.01% of jobs), we are able to fit our 2019 data to Pareto distributions with $\alpha = 0.69$ (CPU) and $\alpha = 0.72$ (memory) with an R^2 goodness of fit of over 99% in both cases.

Pareto distributions, particularly those with $\alpha < 1$, exhibit a strong heavy-tailed property, where a small number of the largest jobs comprise most of the load [9]. The "heavy-tailed property" is far more extreme than the commonly cited "80-20 rule," where the 20% largest jobs comprise 80% of the load. In prior empirical studies of compute consumption and file sizes [9, 17–20], the authors observe a heavy-tailed property where the 1% largest jobs comprise 50% of the load. The heavy-tailed property we observed in the 2019 traces is even more extreme: the largest 1% of jobs comprise 99.2% of the CPU load (99.1% of the memory load), and the largest 0.1% of jobs comprise 93.1% (92.6%) of the load.

We call the largest 1% of jobs *hogs*, and the remaining 99% of jobs *mice*. A later section discusses some of the consequences of this for workload scheduling.

7.1 Comparison with 2011 data

We see a similar story, albeit not quite so extreme, for the 2011 data (see Table 2 and Figure 12). The 2011 data is a factor of eight smaller in scale than 2019, and the raw machine sizes were different. However, we can still directly compare the squared coefficient of variation and the overall distribution parameters, both of which are invariant to normalization. Additionally, both 2011 and 2019 resource usage values are measured in terms of the fraction of available resources at the time, which means that the results have a similar interpretation.

The 2011 data is not that different from the 2019 data: both sets of data follow Pareto(α) distributions (see Table 2), for both CPU and memory. The 2011 data is somewhat less variable (C^2 values for both CPU and memory usage are about 4 times lower than in 2019)[1] and somewhat less heavy-tailed in terms of the fraction of load made up by the largest 1% and 0.01% of jobs – but still very high compared to other published data. Our general characterization of jobs into *hogs* and *mice* appears to hold consistently across the years.

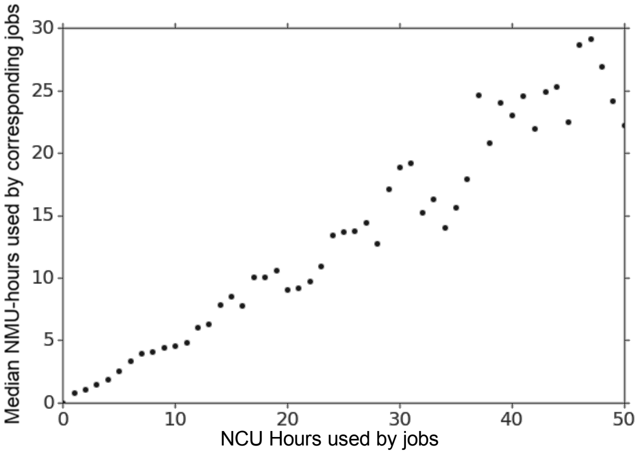


Figure 13. Correlation between compute and median memory consumption for jobs grouped into 1-hour NCU buckets.

We note that [27] claimed that Google compute consumption and memory consumption did *not* follow power-law distributions in 2011. This is because their analysis looked at instantaneous job sizes, not the integral of consumption over time, so we are looking at different signals.

7.2 Correlations between compute and memory consumption

Given that compute consumption and memory consumption follow almost the same distribution (see Figure 12), it is reasonable to ask whether these metrics are correlated. Our analysis suggests they are. Figure 13 shows NCU-hours on the x-axis, quantized into buckets of size 1 NCU-hour. For each of those buckets of size 1 NCU-hour, we plot (on the y-axis) the median NMU-hours consumed by jobs that fall within that bucket. The result is almost a straight line (with Pearson correlation coefficient of 0.97): the higher the NCU-hours used, the higher the median NMU-hours. This is not entirely surprising as the job duration is a common factor of both metrics.

[1] Although the C^2 values are lower for the 2011 data, both the mean and variance are higher for the 2011 data. This is consistent with the fact that the 2011 CCDF curves in Figure 12 stochastically dominates the corresponding 2019 CCDF curves.

7.3 Implications of compute and memory consumption for queueing delay and scheduling

The above results regarding compute and memory consumption have many implications for queueing delay and scheduling. Our goal is not to solve the scheduling question here, but simply to explain the implications of our measurements. It is well known that mean queueing delay is directly proportional to the squared coefficient of variation (C^2) of job service requirements [18]. Specifically, the Pollaczek-Kinchin formula [26] tells us that, for an M/G/1 queue:

$$\mathbb{E}[\text{queueing delay}] = \frac{\rho}{1-\rho} \cdot \frac{C^2+1}{2},$$

where ρ is the load and C^2 is the squared coefficient of variation of job sizes.

Observe that high C^2 means that we can expect high queueing delay even when the system load is low. The intuition is that when C^2 is high, there is a wide range of job sizes, and inevitably we will end up with situations where small jobs (the mice) get stuck waiting behind large jobs (hogs). Scheduling in large scale data-centers is far more complex than an M/G/1 queue, given that there are hundreds of thousands of servers, and jobs themselves occupy many servers at a time. However, the basic message still holds: when C^2 is high, as it is at Google, one must find a way to schedule jobs so that the mice are either prioritized over the hogs or are isolated from them in some other way.

If the scheduler were to ensure that just 1% of the jobs (the compute hogs) did not get in the way of the other 99% of the jobs, the latter could see little to no queueing, and be run much more quickly. The strong correlation between compute and memory consumption means that the compute hogs are also likely to be the memory hogs so that we can just think about how to generally move "hogs" out of the way of the remaining 99% of jobs, rather than handling CPU and memory separately.

8 Vertical scaling

Borg now supports an automatic resource scaling system called Autopilot, which is described in detail in a companion paper [29]. Autopilot supports 3 vertical scaling strategies: *not autoscaled*, *fully autoscaled*, or *autoscaled with constraints* and these are captured in the 2019 trace [37]. As in Kubernetes [14], the term "vertical autoscaling" is used to describe the ability of the system to automatically adjust (increase or decrease) the number of resources assigned to a task.

Autopilot aims to remove the burden of specifying a job's resource requirements. This can be challenging because asking for too few resources can be catastrophic: a job might fail to meet user service deadlines or even crash. For this reason, most users request as many resources as they think they might need. Autopilot instead strives to improve efficiency by reducing *slack* resources as far as possible, i.e., the gap between the requested resources and those actually used. This translates directly into efficiency improvements and costs savings, by allowing more work into the system.

Autopilot makes use of historical data from previous runs of the same or similar jobs in order to configure the initial resource request, and then continually adjusts the resource limits as the job executes so as to minimize slack. In order to give an idea of how well Autopilot works, we looked at how the slack varied depending on the auto-scaling strategy chosen. The precise metric we chose to use is the *peak NCU slack*, which is defined as the gap between the per-task limit and the peak (maximum) observed usage in a 5-minute sample:

$$\text{peak NCU slack} = \frac{\max(0, \text{allocated NCU} - \text{max NCU})}{\text{allocated NCU}}.$$

We compute this for every sampled (5 minute) data point for every task, and show the resulting CCDF in Figure 14. From this we can clearly see that both of the auto-scaling techniques are better than manual configuration; and further that *unconstrained autoscaling* is the clear winner, reducing the peak NCU slack by more than 25% for the vast majority of jobs. For a more thorough analysis of the effectiveness of Autopilot, please refer to [29].

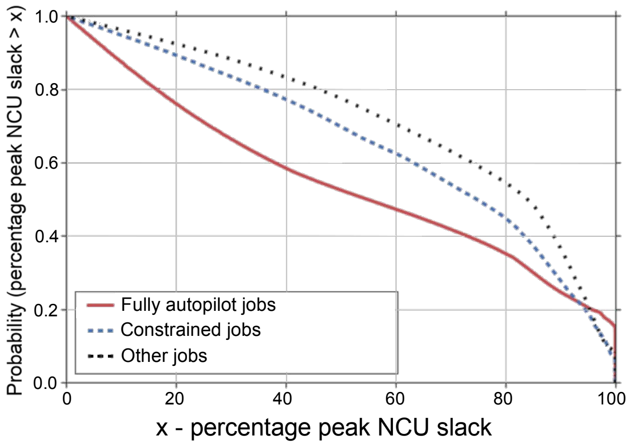


Figure 14. CCDF of the peak NCU slack (limit minus peak usage as a fraction) for fully autoscaled jobs, jobs that are autoscaled but under vertical scaling constraints, and jobs that are not autoscaled.

9 Lessons learned from trace generation

The Borg system generates huge quantities of monitoring data. Converting that into "only" 2.8 TiB of data that made sense to external researchers was a non-trivial challenge. This section describes some of the steps we took.

Exposing enough details without losing focus. Borg gives its users great freedom to express their requirements, and a huge variety of configuration and tuning patterns have resulted. To support its users and administrators, Borg has evolved sophisticated monitoring and bookkeeping systems, which generate huge amounts of monitoring data – not all of which is internally consistent. We found it necessary to clean and extrapolate data to fit an abstract mental model understandable by people outside Google. To support this, we extended a distributed Flume pipeline that had originally been built for the 2011 trace generation process.

Providing explanations for unreasonable cases. We found it helpful to check a raft of logical invariants such as "the total resource usage of all instances on a machine should be smaller than the machine's capacity"; or "a SUBMIT event should happen before any termination event". Given the vagaries of large-scale trace data collection, we found that most of these invariants were violated occasionally. Even so, they proved invaluable in giving us confidence in the trace data.

Automated validation. To support such validations, we evolved from a system of one-off scripts to a repeatable pipeline that could be applied automatically. In retrospect, we should have started with that: at this scale, paranoia is a helpful default.

Using BigQuery. It was empowering to be able to execute near-arbitrary queries against a multi-GiB dataset in a few tens of seconds. This made our validation work easier and analyses simpler. We hope that users of the new trace will share our enthusiasm.

10 Research directions

This paper reports on the first public analyses of the new 2019 Google trace data set – but this is just the tip of the iceberg of the opportunities it makes available. What follows are a few suggestions for research questions that we wished that we had time for before the publication deadline, but which others may want to explore.

1. **Explainable scheduling:** schedulers are highly complex entities that have to account for rapid changes in both the state of the cluster and the workload. It would be nice to be able to provide explanations for *why* the scheduler made the decisions it made – either to help system operators understand what is going on (or is about to), or to provide guidance to end users on how they could better use the cluster.
2. **How far can overcommittment be pushed?** It is clear that statistical multiplexing is important in achieving high utilization – but it comes at a cost. How far can it be pushed, under what circumstances?
3. **Gang scheduling:** Borg will start a job as soon as any of its tasks are running. (Users can ask to wait for more tasks to be running, but few do.) Its algorithms are generally relatively simple greedy heuristics. How could we do better?
4. **Why is the average utilization relatively low?** Despite a great deal of effort, the average utilization of compute and memory resources is still relatively low. Why is that? One hypothesis is that it is related to disaster recovery protocols, in which a third of the external load (in a 2+1 redundancy system) can be switched to a target cell nearly instantaneously. Another one is that there are temporal and/or per-machine variations that prevent tighter bin-packing. What other explanations are there? Can average utilization be further increased?
5. **Scheduling to combat heavy tails:** we've seen that the top 1% most compute-intensive jobs (the "hogs") comprise over 99% of the total CPU usage. There is interesting research to be done on how to schedule jobs in a way that allows the remaining 99% of jobs (the "mice") to have partial or full isolation from these hogs, so that they can experience what appears to be a very lightly loaded environment.
6. **Inter-cell variations:** the analysis here has barely scratched the surface of differences between the 8 cells in the new traces. We expect there is much to be learned by more detailed studies that compare their behaviors.

11 Related work

Traces have been around a long time, and long before Borg. We do not have space to present a thorough survey of this space, and so focus here on examples of more recent compute cluster traces and workloads.

An early 7-hour long precursor to the Borg traces [21] was analyzed in [25]. [27] offered a more thorough analysis of the original 2011 trace, highlighting the heterogeneity observed in the workload. Since then, several hundred researchers have used the 2011 trace or written about them. Google Scholar claims there have been 555 citations for the 2011 trace format document by early March 2020 – a tiny subset is listed in [7].

Several other companies have released cluster traces. An analysis of Microsoft's Azure VM workloads [8] concludes that certain VM behaviours are consistent over time, and historical data can be used as an accurate predictor for future behaviour. From this observation they create a resource predictor, which collects VM data, learns behaviours offline and provides predictions to resource managers; this is analogous to Borg's Autopilot (Section 8). Their paper is accompanied by a publicly available trace [3] containing traces collected in 2017 and 2019, which contains information about 2 M virtual machines and 2.6 M virtual machines respectively across a 30-day window. Both traces are well-cited within the research community.

Alibaba has released two traces [1]: the 2017 version contains information from 1300 machines over a period of 12 hours, while the 2018 version contains information for about 4000 machines over a period of 8 days. [33] includes an analysis of task dependencies from this data.

The Parallel Workloads Archive [12] contains HPC traces collected between 1993 and 2015, and the Grid Workload Archive [15] contains anonymized workload traces from scientific grid environments.

12 Conclusions

Understanding large-scale compute cluster workloads is of ever-growing importance as we come to rely more and more on the capabilities delivered by "warehouse-scale computers". Traces of such systems are important to the academic community as demonstrated by the hundreds of published studies that have built on or used the 2011 trace.

This paper describes the next generation of Borg traces and provides a preliminary analysis, including longitudinal comparisons with the 2011 trace. Many of the new features described in the Borg paper [35] are reflected in the new trace, as well as several new kinds of information.

We found that although the cluster cell sizes are relatively unchanged, the workload has increased substantially in both an absolute sense, and in terms of the scheduler load, as has the effective utilization of the machines. Consistent with [35], we see big variations in the behaviors across different cells. Much of the previous "free" tier work has migrated to the best-effort batch tier, and the disparity between the resource consumption of the big jobs ("hogs") and the small ones ("mice") is now more extreme than in any other reported trace: the largest 1% of jobs now represent 99% of both compute and memory resource consumption, with a squared coefficient of variation over 23 000. Finally, we observe that today's resource prediction algorithms appear more efficient than users manually defining their own resource requirements.

We hope that others will find the new traces as useful as the 2011 ones seem to have been, and look forward to even more exciting ideas and analyses being generated now that they are available.

Acknowledgments

We thank numerous engineers who work on Borg and the monitoring infrastructure. We also want to thank Rohit Jnagal, Krzysiek Rzadca, the EuroSys reviewers, and our shepherd, Valerie Issarny, for their feedback on this paper.

References

[1] Alibaba cluster data: using 270 GB of open source data to understand Alibaba data centers. Blog post, url = https://www.alibabacloud.com/blog/594340, Jan. 2019.

[2] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben. On the diversity of cluster workloads and its impact on research results. In *USENIX Annual Technical Conference (USENIX ATC)*, pages 533–546, Boston, MA, USA, July 2018. USENIX Association.

[3] Azure Public Dataset. https://github.com/Azure/AzurePublicDataset. Accesses 2020-03.

[4] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter. Adaptsize: Orchestrating the hot object memory cache in a content delivery network. In *14th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 483–498, Boston, MA, USA, 2017. USENIX Association.

[5] Google BigQuery. Online documentation, https://cloud.google.com/bigquery/., 2020. Accessed 2020-03.

[6] Cumulative distribution function, Accessed Nov. 2019. https://en.wikipedia.org/wiki/Cumulative_distribution_function.

[7] Google cluster data bibliography, Accessed Nov. 2019. https://github.com/google/cluster-data/blob/master/bibliography.bib.

[8] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *26th Symposium on Operating Systems Principles (SOSP)*, pages 153–167, Shanghai, China, 2017. ACM.

[9] M. Crovella. Performance evaluation with heavy tailed distributions. In *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 1–10, London, UK, 2001. Springer-Verlag.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *6th Conference on Symposium on Operating Systems Design & Implementation (OSDI)*, pages 10–10, San Francisco, CA, USA, 2004. USENIX Association.

[11] N. El-Sayed, H. Zhu, and B. Schroeder. Learning from failure across multiple clusters: a trace-driven approach to understanding, predicting, and mitigating job terminations. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 1333–1344, June 2017.

[12] D. Feitelson, D. Tsafrir, and D. Krakov. Experience with the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74, 10 2014.

[13] C. N. C. Foundation. Kubernetes. http://k8s.io; accessed 2019-11.

[14] C. N. C. Foundation. Vertical pod autoscaler. https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler; accessed 2019-11.

[15] The Grid Workloads Archive. Online repository, http://gwa.ewi.tudelft.nl/.

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

[17] M. Harchol-Balter. The effect of heavy-tailed job size distributions on computer system design. In *ASA-IMS Conf. on Applications of Heavy Tailed Distributions in Economics*, 1999.

[18] M. Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action.* Cambridge University Press, New York, NY, USA, 1st edition, 2013.

[19] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. In *ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 13–24, Philadelphia, PA, USA, 1996. ACM.

[20] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst.*, 21(2):207–233, May 2003.

[21] J. L. Hellerstein. Google cluster data. Google research blog, Jan. 2010. Posted at http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[22] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: a platform for fine-grained resource sharing in the data center. In *8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*, pages 295–308, Boston, MA, USA, Mar. 2011.

[23] H. Li, D. Groep, and L. Wolters. Workload characteristics of a multi-cluster supercomputer. In *10th International Conference on Job Scheduling Strategies for Parallel Processing (JSSPP'04)*, pages 176–193. Springer Verlag, June 2004.

[24] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of web-scale datasets. In *36th International Conference on Very Large Data Bases (VLDB'10)*, pages 330–339, 2010.

[25] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das. Towards characterizing cloud backend workloads: insights from Google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37(4):34–41, Mar. 2010.

[26] Pollaczek–khinchine formula. Wikipedia, https://en.wikipedia.org/wiki/Pollaczek-Khinchine_formula. Accessed 2020-03.

[27] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *3rd ACM Symposium on Cloud Computing (SoCC'12)*, pages 7:1–7:13, San Jose, CA, USA, 2012. ACM.

[28] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format + schema. Technical report at https://github.com/google/cluster-data, Google, Mountain View, CA, USA, Nov. 2011. Revised 2014-11-17 for version 2.1.

[29] K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand, and J. Wilkes. Autopilot: workload autoscaling at Google. In *15th European Conference on Computer Systems (EuroSys'20)*, Heraklion, Crete, Greece, 2020. ACM.

[30] B. Schroeder and M. Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. *Cluster Computing*, 7(2):151–161, Apr. 2004.

[31] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *8th ACM European Conference on Computer Systems (EuroSys'13)*, pages 351–364. ACM, 2013.

[32] A. Sîrbu and O. Babaoglu. Towards data-driven autonomics in data centers. In *International Conference on Cloud and Autonomic Computing (ICCAC)*, Cambridge, MA, USA, Sept. 2015. IEEE Computer Society.

[33] H. Tian, Y. Zheng, Y. Zheng, W. Wang, and W. Wang. Characterizing and synthesizing task dependencies of data-parallel jobs in Alibaba Cloud. In *10th ACM Symposium on Cloud Computing (SoCC'19)*, Chaminade, Santa Cruz, CA, US, Nov. 2019. ACM.

[34] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *4th Annual Symposium on Cloud Computing (SoCC'13)*, pages 5:1–5:16, Santa Clara, CA, USA, Oct. 2013.

[35] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *10th European Conference on Computer Systems (EuroSys'15)*, pages 18:1–18:17, Bordeaux, France, 2015. ACM.

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

[37] J. Wilkes. Google cluster-usage traces v3. Technical report at https://github.com/google/cluster-data, Google, Mountain View, CA, USA, Nov. 2019.