Towards Understanding HPC Users and Systems: A NERSC Case Study

Gonzalo P. Rodrigo[a,1], P-O Östberg[a], Erik Elmroth[a], Katie Antypas[b], Richard Gerber[b], Lavanya Ramakrishnan[b]

[a] *Dept. Computing Science, Umeå University SE-901 87, Umeå, Sweden*
[b] *Lawrence Berkeley National Lab Berkeley, CA 94720, USA*

Abstract

The high performance computing (HPC) scheduling landscape is changing. Previously dominated by tightly coupled MPI jobs, HPC workloads are increasingly including high-throughput, data-intensive, and stream-processing applications. As a consequence, workloads are becoming more diverse at both application and job level, posing new challenges to classical HPC schedulers. There is a need to understand the current HPC workloads and their evolution towards the future in order to perform informed scheduling research and enable efficient scheduling in future HPC systems.

In this paper, we present a methodology to characterize workloads and asses their heterogeneity, both for a particular time period and as they evolve over time. We apply this methodology to the workloads of three systems (Hopper, Edison, and Carver) at the National Energy Research Scientific Computing Center (NERSC). We present the resulting characterization of jobs, queues, heterogeneity, and performance that includes detailed information of a year of workload (2014) and evolution through the systems' lifetime. Among the results, we highlight the observation of discontinuities in the jobs' wait time for priority groups with high job diversity. Finally, we conclude by summarizing our analysis to establish a reference and inform future scheduling research.

Keywords: workload analysis, supercomputer, HPC, scheduling, NERSC, heterogeneity, k-means

1. Introduction

High performance computing (HPC) supports scientific research by providing capacity to run large simulations or solve large mathematical problems. Such applications largely rely on the tightly coupled MPI model, which, as a consequence, has dominated HPC workloads. However, the workload configuration is changing as HPC systems' use evolves. For instance, some scientific fields like biology or astrophysics increasing rely on analysis of large datasets. Also, as compute capacity keeps growing, simulations produce larger datasets that require analysis. Finally, semiconductor advances enable real experiments to produce higher resolution data that requires processing. Pushed by these use-cases, workloads are becoming more diverse, increasing the importance of high-throughput, data-intensive, and stream-processing applications. These applications differ in performance models and target objectives from the classical parallel tightly coupled. As a consequence, current HPC schedulers might not produce optimal decisions since they support diverse workloads which configurations differ from what batch schedulers were designed to support (uniform and MPI dominated).

Supporting the new workload landscape and its future evolution requires new scheduling models that need to be investigated. Such research must be informed by a characterization of the state, with a focus on diversity, of current workloads in HPC centers and their evolution. However, existing work on workload modeling [1] characterizes systems that are too old, too small, or not representative of the top HPC systems. Also, previous work did not focus on the workload diversity, a new trait present in recent workloads. Thus, there is a need to investigate the workloads in current HPC centers to understand users and applications requirements and project them in the future.

In this work, we present a methodology to characterize HPC workloads in detail. It includes classical workload analysis methods such as value distribution analysis on job variables (e.g., degree of parallelism or runtime), system utilization estimation, or overall wait time analysis. However, it includes an innovative method to analyze job geometry (allocated resources and runtime) diversity. This method employs k-means clustering to identify dominating groups of jobs in the workload according to their geometry (runtime and allocated CPU cores). Under this analysis, workloads with more job geometry clusters are considered more diverse, and vice versa. Also, job groups can be mapped on the waiting queues (priority categories) to analyze jobs diversity within each queue. Consequently, an analysis of the correlation of queue diversity and wait times for jobs of different sizes is performed. This analysis verifies that jobs' wait time match the expected values according to priority and system configuration (i.e., larger jobs should wait longer, higher priority jobs should wait

Email addresses: gonzalo@cs.umu.se (Gonzalo P. Rodrigo), p-o@cs.umu.se (P-O Östberg), elmroth@cs.umu.se (Erik Elmroth), kantypas@lbl.gov (Katie Antypas), ragerber@lbl.gov (Richard Gerber), lramakrishnan@lbl.gov (Lavanya Ramakrishnan)

[1] Work performed in part at the Lawrence Berkeley National Lab.

shorter) or if they deviate due to job heterogeneity.

We apply this methodology to the workloads of three systems (Hopper, Carver, and Edison) at the National Energy Research Scientific Computing Center (NERSC) [2]. The output of these analyses is a reference of the workloads of three systems representative of others at the HPC community: Carver is a terascale IBM high performance cluster built on commodity hardware supported by an Infiniband interconnect; Hopper is an early petascale Cray supercomputer based on AMD processors; and Edison is a more modern and energy efficient petascale Cray supercomputer based on Intel processors. The results include a detailed analysis on the jobs, queues, and system behavior in each year over their lifetime for the Hopper and Carver and in 2014's for Edison. Yearly data of Hopper and Carver is compared to produce a trend analysis that allows to observe the evolution of their lifetime. These results establish a first data-point to predict future HPC workloads to designing future resource management models.

Our workload analysis methodology can also be used to support informed short-term and long-term decisions at HPC centers. Periodical workload analyses can be aggregated in a growing trend analysis that can reveal changes in the user behavior and system performance. Also, the boundary geometries (i.e. runtime and degree of parallelization) in workload's job clusters might be used as a starting template to define priority groups (queues) to avoid mixed queues and minimize discontinuities in expected job wait time behavior.

Specifically, in this paper:

- We propose analysis methods to understand and compare the workload diversity: how self similar are jobs in the workload and their mapping on the prioritization queues.
- We define a method to analyze the wait time of jobs depending on their geometry, queue priority, and diversity.
- We provide a detailed job, queue, performance, and diversity characterization of the NERSC workload, and their evolution over time. The results allow to understand the users, system behavior, and the effect of queue heterogeneity on jobs' wait time.
- We present a summary of analysis results and compare them with characterizations of other existing HPC workloads.

The rest of the paper is organized as follows. We present background on HPC systems, scheduling, and workload analysis in Section 2. A high level description of our method and the analyzed systems is presented in Section 3. The details of the methodology and its application to the NERSC workloads are described in Sections 4 to 7. Finally, we provide a summary of our results together with conclusions in Section 8.

This work includes and extends previously published work from the same authors [3], [4].

2. Background

This section describes the challenges in the HPC community that motivate this work and presents background on parallel job scheduling and workload analysis relevant to understand our methodology and results.

2.1. Challenges in HPC scheduling

The challenges of resource management in HPC are changing. New application characteristics and technological shifts are bringing new concepts and requirements to the scheduling models and system architectures. In this section, we highlight some workloads' changes that stress the importance of our analysis methods.

Stream applications are becoming more present in HPC systems. Scientists conduct experiments that would benefit from real-time processing of large amounts of data on HPC systems (e.g. X-Ray Micro-diffraction on Advanced Light Source at LBNL [5]). Real-time processing could potentially be performed by providing resources through advance reservations. Advance reservations, however, have a negative impact on the overall utilization, showing the need for real-time scheduling (i.e. low-latency allocation of resources, with no previous reservation as a response to a real-time event). As another step in application evolution, scientific experiments in fields like biology, earth sciences, or high energy physics are increasingly relying on data analysis to extract useful information from large experimental datasets, or results from large simulations [6], [7]. These applications increase the importance of data-intensive computational models in HPC workloads, or the composition of different applications through workflows (e.g., simulation followed by results analysis). These changes motivate us to analyze the workloads at supercomputers to understand their current characteristics.

The importance of stream and data intensive applications point at an increasing diversity in workloads not only dominated by large tightly coupled parallel jobs. Diversity might affect the performance of the scheduler, which governs the execution of applications in HPC systems. For example, the impact of the scheduling decisions is different across applications: e.g. delaying one job belonging to a workflow may have a significant impact on its overall run time, while delaying a stream job that has to be rapidly scheduled might render it useless. Also, schedulers are unaware of the different architecture-related constraints in applications (e.g. I/O bound performance, loosely coupled jobs, and data locality). However, information about such constraints is required for the scheduler to perform optimal placement decisions to maximize the applications' performance. Understanding the impact of the application diversity on the system motivates our workload heterogeneity analysis.

2.2. Scheduling

HPC schedulers optimize job placement to achieve the highest system utilization possible with a reasonable turn-

System	Vendor	Model	Built	Nodes	Cores/N	Cores	Memory	Network	TFlops/s	Service
Hopper	Cray	XE6	2010	6,384	24	154,216	212 TB	Gemini	1280	Jan'10
Edison	Cray	XC30	2013	5,576	24	133,824	357 TB	Aries	2570	Jan'13
Carver	IBM	iDataPlex	2010	1,120	8/12/32	9,984	147 TB	Infiniband	106.5	Apr'10

Table 1: Edison, Hopper, and Carver characteristics

around time according to the job priority. The most common base technique in schedulers is FCFS (First-Come, First-Served) [8]. With FCFS, jobs are selected in order, reserving the associated resources required for a job. However, with FCFS, the scheduler has to drain the system in order to schedule a large job, leading to resource fragmentation that reduces the overall utilization. Thus, backfilling is normally used to move jobs forward to fill resource gaps produced by the FCFS. Backfilling provides an ordered search in the waiting queue to map jobs to empty resource windows even if they are not at the head of the queue [9].

The quality of the results of the backfilling algorithm depends on the user's wall clock time estimation [8]. If a job wall clock time is overestimated, the scheduler will assign an unnecessary large resource window, reducing the opportunities to schedule a job through backfilling. On the contrary, if wall clock time is underestimated (i.e., runs over its limit), the system will kill the job resulting in lost work. These effects motivate the jobs wall clock time accuracy (relationship between estimated and actual wall clock time) characterization presented in Section 4.2.

Finally, a job's turnaround time depends on its priority (influencing its progress on the scheduler wait queue in each scheduling pass), geometry (jobs requiring more resources are harder to schedule), and requested resource load (how many jobs compete for the same resources). However, job diversity in the queues might affect this relationship. In Section 6.2 we present an analysis of the possible impact of these factors (including job diversity) on the job's wait time.

2.3. Related work on workload analysis

Previous work on scientific Grid and HPC workloads characterization is found in the Grid Workloads Archive [10] and the Parallel Workload Archive [1]. The archives contain job and performance characteristics (run time, parallelism, inter-arrival time, wait time, disk space, and memory), but their analyses overlook the workload heterogeneity. Also, analyzed systems are either at least 10 years old or significantly smaller than the current top HPC systems. Our work extends them by addressing jobs' heterogeneity and performing analyses on large, more recent systems (e.g. Edison was deployed in 2014 and still ranks 60 in the Top 500 list in February 2017).

Job heterogeneity has been observed in industrial workload diversity analysis [11], which introduces k-means as a tool for job similarity clustering. This work inspired our workload diversity analysis method for HPC workloads, which we complemented with per queue analysis and a new methodology to compare the degree of heterogeneity across systems and system states.

A previous analysis [12] on the applications run on Hopper in 2012 characterizes the importance of the different applications run on the system, with an initial insight on the jobs' geometry and memory requirements.

3. Methodology

In this section we describe the three systems analyzed (their characteristics, workload, scheduling model, and configuration), our data source (size, time span, format), analysis framework (motivation for analyzed variables), and trend analysis methodology.

3.1. System descriptions

In this work, the results of characterizing the workload of three HPC systems are presented. In this section, we describe the characteristics of these systems to enable the discussion about the applicability of our results to other systems.

3.1.1. System characteristics

NERSC is a HPC center at Lawrence Berkeley National Lab, that has the mission to provide computing infrastructure and tools for scientists performing research of relevance to the DOE (Department of Energy). Our work analyzes the various years of real jobs from three of NERSC's systems: Carver, Hopper, and Edison. These three systems were selected because their different hardware characteristics and origin in the timeline of HPC systems evolution, which can be observed in Table 1. Carver is a terascale IBM iDataPlex Linux cluster [13] deployed in April 2010. Its configuration is the closest to commodity hardware servers of the three systems and it is supported by an Infiniband interconnect. Hopper and Edison are specialized Cray supercomputers with custom interconnects [14]. Hopper is a petascale Cray XE system, based on AMD processors and a Gemini interconnect, and deployed in 2010. Edison is a newer, more power efficient petascale Cray XC30, constructed with Intel processors supported by a Aries interconnect and deployed in 2014. Thus, these systems allow us to capture the workload characteristics of high-end clusters and supercomputers, belonging to different HPC system generations and optimized for slightly different applications.

On the resource management side, all three systems use the Moab scheduler [15, 13, 16] running atop the Torque resource manager [17]. Edison's workload manager was replaced by Slurm at the end of 2015.

3.1.2. Workload

Over 5000 users and 700 distinct projects use NERSC resources [18, 12]. The workload is composed of applications from various scientific fields like Fusion, Chemistry, Material Science, Climate Research, Lattice Gauge Theory, Accelerator Physics, Astrophysics, Life Sciences, and Nuclear Physics.

In addition to serving typical MPI workloads, Carver provides a *serial* queue [19]. The serial queue allows users to submit and execute jobs with a very low degree of parallelism (i.e., one single core). Carver has 80 compute nodes allocated to serial jobs. Serial queues were added to Hopper and Edison in late 2014. The serial queue on Edison and Hopper is configured via a super-job run under the special Cluster Compatibility Mode (CCM). There are a total of 15 compute nodes each allocated to run serial jobs on Edison and on Hopper. Serial queues contain jobs running long time (limited to 48 hours) on a single core. The purpose of the serial queue is to increase resource utilization density. It serves packs jobs on the same node that do not benefit from parallelism and which performance is either not critical or rarely affected by resource sharing.

The conclusions of this work are only based on the job related information of the workload. Run time characteristics of applications, execution schema or other variables were not considered or analyzed in this study.

3.1.3. Scheduler characteristics

The configuration of a system scheduler has an impact on the system performance (i.e., utilization, wait time) and the workload shape: e.g., jobs allocation sizes will cluster around the allowed values in submission queues. In this subsection, we present the configuration of the analyze systems scheduler to provide context for later analyses.

First, node sharing is only enabled for nodes executing jobs from the *serial* queue to avoid performance degradation [20]. In order to keep the same baseline, we consider *cores* as the degree of parallelism unit in our analysis.

In all systems there is a distinction between the queues chosen at submission time (Torque) and the queues that the scheduler use for priority calculation (Moab). Users submit jobs to the Torque *submission queues*. Moab has its own queue configuration - the *execution queues*. Torque translates the queue into Moab's execution queues and passes the job to the scheduler. Submission queues can be mapped to a single or multiple execution queues. For example, jobs of up to 10 hours of runtime maybe submitted to the same submission queue, to be sorted into two execution queues with ranges of $[0, 5)$, $[5, 10)$ runtime hours. Table 2 shows queue properties that govern the scheduling decisions for our three systems. The properties are explained below:

Maximum wall clock time (Torque): Each queue has an upper limit for a job's estimated wall clock time specified by the user at submission time. If a job's estimated wall clock time is longer than this limit, submission fails. If a job runs longer than the user estimated wall clock time, the job is terminated.

Number of cores (Torque): Each queue has a predefined minimum and maximum limit of a job's requested number of cores. Submission of a job allocating a number of cores outside this range will fail.

Queue priority (P) (Moab): Each queue in the system is assigned a priority (represented as an integer where a higher number represent a higher priority).

Eligible jobs limit per user (E) (Moab): Only the first E jobs of the same user in the same execution queue are eligible for scheduling. This can affect a job's wait time. For example, if a user would submit 25 jobs to the serial queue on Carver, only the first 20 jobs will be considered for scheduling. The last five jobs will only be considered to be scheduled after the first five jobs have finished. This can impact wait times for the jobs where the last five jobs may have significantly higher wait times than the other 20 jobs.

The execution queues do not exist as separate data structures inside Moab. All jobs are stored in a single queue. When a job is passed to Moab, it is inserted in its job waiting queue with a job priority of zero. In every scheduling pass, the job priority is recalculated by adding a value, which depends on the associated execution queue priority. If a job is in a higher priority queue, the job priority will grow faster and it will be eligible for execution more quickly. The analysis of the impact of the queues' characteristics on jobs wait time is presented in Section 6.2.

3.1.4. Queues configuration

The analyzed system's scheduler re-calculates the jobs priority depending on the queue they are submitted to. Since the configuration of such queues affect the overall system behavior, we present their configuration in detail in this section.

Table 2 presents the execution queue configuration of Edison, Hopper, and Carver used in the analysis. It covers each queue's job maximum run time (Wall Clock Time), job allowed allocations in numbers of cores (Cores), number of eligible jobs allowed to be scheduled simultaneously (E), and the priority of the queue (P). This information allows us to understand the reasons for different wait time behaviors between queues.

The batch queue policies influence the jobs execution order. These policies changed slightly through the studied period. To simplify the analysis, we used the settings that were most common through the period of study. Also, some queues were filtered out of this study as they represented too little of the workload, or were related to system maintenance or tests.

Edison and Hopper map their queues on a single set of resources (independent for each system). However, Carver queues are mapped in sets that partly overlap: general set (1080 nodes), *matgen* set (64 nodes, subset of the general set), *xlmem* set (two nodes with large memory capacity), and *serial* set (80 nodes, not overlapping). Different

Hopper Queues	Wall clock	Cores	E.	P.	**Edison** Queues	Wall clock	Cores	E.	P.	**Carver** Queues	Wall clock	Cores	E.	P.
bigmem	24h.	1-8,856	1	0						matgen_low	Unk	1-256	66	0
ccm_int	30m.	1-12,288	2	1	cm_int	30m.	1-12,288	2	1	matgen_prior	Unk	1-256	66	10
ccm_queue	96h.	1-12,288	16	1	ccm_queue	96h.	1-16,368	16	0	matgen_reg	Unk	1-256	66	1
debug	30m.	1-12,288	2	1	debug	30m.	1-12,288	2	1	debug	30m.	1-256	1	2
low	48h.	1-16,392	6	-3	low	48h.	1-16,392	6	-3	low	24h.	1-256	3	-2
premium	48h.	1-49,152	1	2	premium	36h.	1-49,152	1	2	xlmem_sm	72h.	8	1	0
reg_1hour	1h.	Unk.	8	0	reg_1hour	1h.	Unk.	16	0	xlmem_lg	72h.	32	2	0
reg_big	36h.	49,153-98,304	2	1	reg_big	36h.	49,153-98,304	2	1	reg_big	24h.	257-512	1	0
reg_long	96h.	1-1,536	4	0						reg_long	168h.	1-128	1	0
reg_med	36h.	16,369-49,152	4	1	reg_med	36h.	16,369-49,152	8	1	reg_med	36h.	129-256	2	0
reg_short	6h.	1-16,368	16	0	reg_short	6h.	1-16,368	24	0	reg_short	4h.	1-128	4	0
reg_small	48h.	1-16,368	16	0	reg_small	48h.	1-16,368	24	0	reg_small	48h.	1-128	3	0
reg_xbig	12h.	98,305-146,400	2	0	reg_xbig	12h.	98,305-131,088	2	1	reg_ xlong	504h.	1-32	1	0
										interactive	30m.	1-64	1	2
thruput	168h.	1-48	500	0	killable	48h.	1-16,368	8	0	serial	48h.	1	20	-

Table 2: Hopper, Edison, and Carver queue characteristics. Jobs have to be within certain limits to be accepted in a queue: requested runtime upper limit (wall clock time) and accepted number cores range (Cores). Eligibility (E.): Maximum number of jobs from the same user in the same queue which are considered in jobs priority recalculation. Priority (P.): Queue priority.

queues have access to different sets: *matgen* queue jobs can only run on *matgen* resources (but jobs from other queues can use them when they are available). *xlmem* nodes can be only used by *xlmem* jobs. This implies that different queues may not present the same ratio of job core-hours requested over resources' core-hours available. How this difference may impact jobs wait time is studied in Section 6.2.

The *serial* queue jobs allocate one core per job and are executed on shared nodes (more than one job per node). Also, this queue has exclusive access to the *serial* resource set so it does not compete with any other queue. Thus, wait times of the *serial* queue should not be compared with other queues.

3.2. Data Source

All workload analysis is performed on the job summary entries from the systems' Torque logs. The data includes 1 year and 1,357,366 jobs for Edison, 4.5 years and 4,326,870 jobs for Hopper, and 4.5 years and 9,508,054 jobs for Carver. The raw data size is 45 GB, which, after filtering and parsing, is reduced to 6 GB of net data.

3.3. Analysis Framework

The analysis framework is composed of a set of scripts that express the data pipeline to process the log data. The developed data pipeline is divided into three parts. The first is the data extractor, which retrieves the log files from the NERSC repository, parses them, eliminates invalid entries and inserts them in a MySQL database. The second component is a Python API to insert, manipulate, and retrieve the data from a MySQL database. The MySQL database is indexed to facilitate the queries based on multiple fields. The third component is the analysis toolkit. It implements the logic to retrieve, analyze, and visualize the data for all the analyses. A specific plotting library was developed to support the graph generation. The code consists of 14K Python lines using the scientific libraries SciPy and NumPy combined with the plotting library MatPlotLib [21]. All analyses were run on an Intel i7 Quad core 8 GB RAM desktop computer. The database is hosted on a department server at Berkeley Lab.

Our analysis focuses on understanding the variables of the workload from the user (i.e., job) and system (i.e., queues and performance) perspectives.

The job perspective includes:

Job size: includes wall clock time, degree of parallelism, and resulting compute time allocation. These parameters define the system boundaries' requirements and job granularity.

Wall clock time accuracy: represents how accurate are the user estimations on the jobs runtime. The variable measures the quality of the information used by the scheduler in its job planning.

Inter-arrival time: models the time between the submission of two jobs. It represents the load to be managed by the scheduler and the overall wait time. For instance, for the same job sizes, a smaller inter-arrival time represents a larger job load.

Job diversity: measures how different the geometries of jobs in the workload are. It includes the analysis of dominant job geometries in the workload.

The queues and their configuration represent the mapping of the prioritization policies to the workload job mix. The study includes:

Queue significance: represents the impact of each queue on the overall system. It allows to understand how the properties of each queue contributes to the overall system behavior according to their importance.

Queue job diversity: analyzes the how self similar are

the jobs within each queue in terms of geometry. This analysis is relevant because execution queues allow the system scheduler to prioritize jobs depending on their geometry. However, the existing queues might not represent the most significant types of jobs (by geometry) present in the job mix. This analysis is focused on two objectives: understanding the diversity of the jobs across the entire workload, and similarity of jobs contained in the same queues.

The performance perspective covers the system utilization and job wait time. Additionally, the job wait time is studied from system, queue, and job geometry points of view. This study allows us to understand the how the effectiveness of the priorities for different job geometries in different queues might be affected by job diversity in the queues.

3.4. Trend analysis

In this work, the workload of Carver and Hopper was analyzed in detail for each year of their lifetime. To simplify interpreation, the detailed analysis is only presented for 2014. However, the relevant results of all years were aggregated in the trend analysis, presenting the evolution of Carver and Hopper workloads through their lifetime. Edison's trend analysis was not performed because not enough workload data was available.

In this work, we focus on understanding the evolution of the system's workload, overall performance, and user behavior. As explained in Section 3.3, workload trend covers the evolution of the job geometry (wall clock time and degree of parallelism). Overall performance is analyzed through the evolution of job wait time. User behavior is analyzed by observing the evolution of the wall clock time accuracy.

Finally, since the trend is performed by analyzing the workload in sequential time periods and aggregating the results over a time line, an adequate period had to be chosen. The size of the workload periods is calculated by detecting repeating user patters in the workloads through Fourier transform analysis on the number of tasks submitted per hour [22]. Results of such analysis are described in Section 7.1 and the dominant detected cycle was one year.

4. Job Characterization

In this section we present the workload analysis of the jobs of Edison, Hooper, and Carver in 2014. This analysis is performed with special attention to job geometry, user submission patterns, and job diversity on all three systems.

4.1. Job geometry

Job's geometry analysis allows to observe the patterns in jobs resource allocation and analyze the job mix that the scheduler manages. All variables are analyzed by calculating their value distribution and consequence Cumulative Distribution Function (CDF). This allows to understand if jobs are dominantly small or large, if theirs sizes concentrate around certain values, or if the job mix includes enough jobs of smaller sizes to allows high system utilization. We follow to present the results for each of the job variables.

Job Distribution	Edison	Hopper	Carver
%Jobs Wall Clock $<$ 2 h.	88%	86%	87%
%Jobs Width $<$ 240 codes	69%	75%	99%
%Jobs Width $\leq$ 1 Node	39%	37%	92%
%Jobs Alloc. $\leq$ 1 core-h.	19%	26%	77%
%Jobs Alloc. $\geq$ 1K core-h.	7%	8%	$\sim$8%

Table 3: Detailed job characteristics distribution analysis

Job wall clock time. Figure 1a, shows the Cumulative Distribution Function (CDF) of the job wall clock time for the three systems in 2014. In the case of Hopper, we observe jobs running up to 160 hours, with a high concentration running under two hours. Table 3 shows an overview of the job characteristics' distribution analysis. It shows that 86-88% of the jobs on all three systems run for less than two hours. For Carver, a large number of the jobs have a wall clock time well under one hour; in fact, 60% of the jobs run for less than 13 minutes.

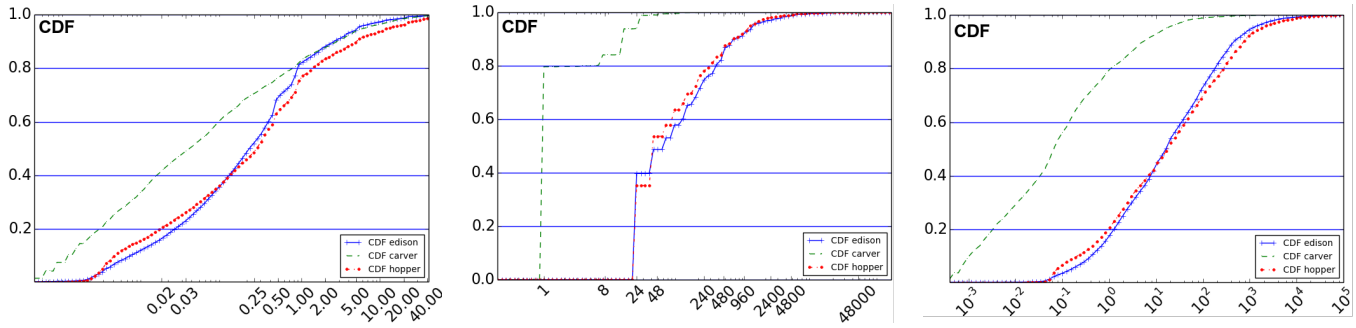
Additionally, all three CDFs present steep slopes around 30 minutes and 6, 12, 24, and 36 hours (better observed in a non-log scale version of the graphs), numbers that are similar to the queues' configured wall clock time limit. These limits are similar across the three machines (more details in Table 2).

Cores per job. Figure 1b presents the distribution of cores allocated to jobs on the three systems. It represents the number of cores requested and allocated to a certain job, and does not include any information on the actual usage of the cores. On Hopper and Edison, requests for a single job range from 24 (1 node) to over 100,000 cores (i.e. close to the full capacity of the systems). A small number of cores are requested for Hopper's jobs: 75% under 240 cores (10 nodes), and 37% of all jobs run on a single node (Table 3). Edison presents a similar pattern with 69% of the jobs running on less than 240 cores and 39% on a single node. Carver shows a different trend from Edison and Hopper. On Carver, many jobs run on a small number of cores: 99% run on 240 cores or less, and 92% of all jobs run on a single node.

Allocated core-hours per job. Figure 1c shows core-hours allocated for the jobs in the system. The figure shows that Hopper and Edison core-hour allocations are similar. Jobs on Hopper and Edison are significantly larger than those on Carver: 99% of Carver jobs individually consume less than one core-hour, in comparison with 42% on Edison and 46% on Hopper. On the other extreme, we can observe that almost 10% of Edison and Hopper jobs individually consume more than 1,000 core-hours.

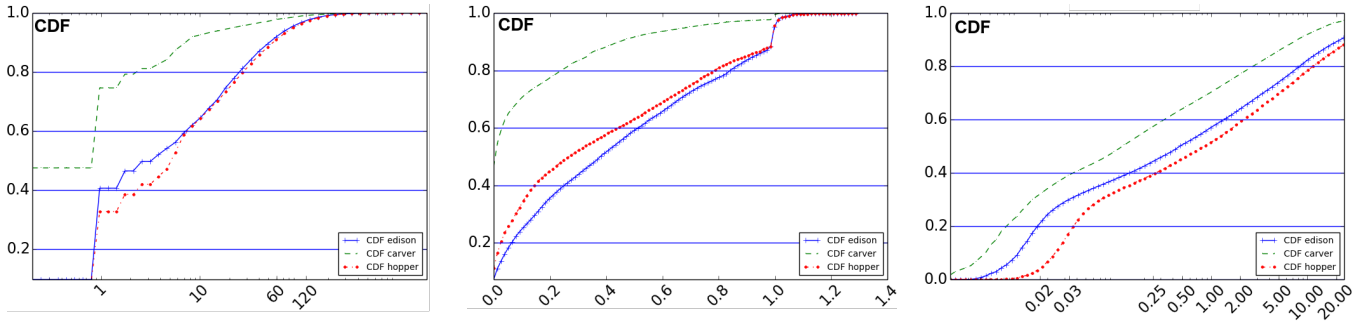
4.2. Job's characteristics

In this section we study other variables of the jobs that depend on other external agents. This includes the user's



(a) Wall clock Time (h.) (b) Cores allocated (c) Core-hours

Figure 1: Job geometry characterization on Hopper, Edison, and Carver. a) Significant percentage (Edison: 87%, Hopper: 82%, Carver: 87%) of the jobs run for 2h or less. b) 69% of Edison, 75% of Hopper and 99% of Carver jobs allocate 240 cores or less. c) Carver's jobs allocate significantly fewer core-hours.



(a) Inter-arrival time (s.) (b) Wall clock time accuracy (c) Wait time (h.)

Figure 2: Job characterization on Hopper, Edison, and Carver. a) Carver receives significantly more job submissions per time unit than the other systems: 40% of jobs are followed by another job within one second. b) 11% of Edison and 10% of Hopper jobs run over the requested time. Carver: 92% of the jobs run under 50% of requested wall clock time. c) Jobs that wait less than 3h to be executed: Edison (67%), Hopper (60%), Carver (79%).

submission pattern (inter-arrival time and wall clock time accuracy) and job's overall wait time (which depends on the scheduler configuration). A more detailed analysis of the job's wait time is presented in Section 6.2.

Inter-arrival time. Figure 2a represents the CDF for the inter-arrival times on Edison, Hopper, and Carver. The inter-arrival time measures the time elapsed between the arrival of consecutive jobs in a system, which can affect the granularity of scheduling and help to understand the load on the schedulers.

Edison and Hopper have very similar distributions: 90% of the jobs have inter-arrival times under two minutes. The remaining 10% are distributed in the 1500-2000 seconds (25-33 minutes) range. On the other hand, more than 95% of Carver's inter-arrival times are under 25 seconds. As the CDFs of the three systems are compared, Carver's inter-arrival times are shorter than those for Edison and Hopper. Thus, when compared to Edison and Hopper, we observe that more jobs are submitted to Carver queues during the studied time period.

Wall clock time accuracy. For each job we study the difference between the actual and the requested wall clock times. The accuracy is defined as $\frac{W}{W^r}$, where W is the actual wall clock time of a job and W_r is the wall clock time that the user requested for the job. The accuracy will be close to one when the estimation is good, and closer to zero when the job running time is overestimated. If the job runs over the requested time, the job will get preempted. However, this is caught during the next scheduling pass. Thus, we see values over one when jobs run over the estimated time.

As discussed in Section 2.2, the wall clock time accuracy affects the backfilling decision quality. Figure 2b presents the distribution of the wall clock time accuracy values for the three systems studied. The initial steep slope of the Carver CDF shows that it executes many jobs that use much less than the requested wall clock time. Edison and Hopper have a more linear CDF for values between zero and close to one. However, in all systems we observe jobs with an accuracy slightly above one (as they exceed their allocated run time and are terminated). We see that the percentage of jobs that run out of wall clock time is higher on Edison (11%) and Hopper (10%) than on Carver (2%). Approximately 60% of Edison and 66% of Hopper jobs run 50% or less of the requested time. On Carver, around 93% of the jobs run 50% or less of the requested

time.

Wait time. Figure 2c presents the distribution of job wait times under 24 hours (jobs with longer wait times are not included in this graph). The figure shows that Hopper has more jobs with longer wait times, followed by Edison and Carver. Considering all the jobs in the system, we see that 61% of Hopper jobs, 67% of Edison jobs, and 80% of Carver jobs have a wait time of less than three hours. Further analysis of the wait time values is presented in Section 6.2.

4.3. Job diversity

The job diversity analysis is based on a machine learning technique and extends a previous work about job grouping within clusters [11]. We construct job geometry tuples that contain job wall clock time and number of cores allocated. Before performing analysis, the tuples are normalized (whiten [23]) to reduce the effect of the value magnitudes on the clustering process. In the analysis, the target is to find the smallest number of k-means clusters [24] among the job geometry tuples where the variation coefficient (standard deviation divided by the mean) is at most 1.1. If jobs are similar, the method will group them in a small number of clusters. Jobs in more diverse workloads are grouped in larger numbers of clusters.

When invoked, k-means produces k clusters from an input dataset and a list of k centroids used as search starting points. However, k-means produce k clusters, not the minimum possible, and does not guarantee that the clusters are not disperse. As a consequence, k-means must be invoked with different k sizes and different start centroids to obtain a minimum possible number of non dispersed clusters. Figure 3 illustrates the algorithm that searches for the minimum clusters in the job geometries, by trying different k values and random centroid points.

It starts by whitening the input job tuples (line 5) to reduce the effect of the value sizes on the clustering. Then, the process to find the minimum number of k-means minimum cluster search is repeated 10 times with different starting centroids. In each trial (lines 7-37), the process starts by producing an initial random centroid set of two points. Then, starting at $k = 2$ the algorithm tries to find k non dispersed clusters, increasing the value of k in each unsuccessful trial (lines 10-37). For each k, k clusters are produced (line 14) and considered *disperse* if their variation coefficient is larger than 1.1 (lines 15-23). Each disperse cluster is divided, i.e., two centroids close to the cluster centroid are produced and added to the obtained cluster lists, increasing the resulting k size in one (line 21). In summary, the obtained clusters are reviewed, if they are not too dispersed, they are left as they are, otherwise they are split, increasing k in one per cluster split. Then, the search is repeated with the new centroids as starting points. The process is repeated until non dispersed clusters are found (lines 26, 27) or the found k is larger than the minimum size of previously observed non dispersed clusters (lines 11,12). The algorithm does not guarantee that the obtained number of clusters is the minimum possible, however it produces a local minimum.

Figure 3: Minimum number of k-means cluster search algorithm for a list of job geometries.

```
1: (repetitions, trialsSmallerK) ← (10, 10)
2: maxCulsterVar ← 1.1
3: minKFound ← −1
4: (finalClust, finalCent) ← (None, None)
5: normJobs ← whiten(allJobs)
6: for i ← 1, repetitions do
7:     seed = genRandomSeed()
8:     k ← 2
9:     cent ← genRandomCentroids(k, seed)
10:    for  j ← 1, trialsSmallerK do
11:        if k >= minKFound and minKFound ≠ −1 then
12:            break
13:        end if
14:        Clust, cent ← kMeans(normJobs, cent)
15:        cvList ← calcCVForClusters(Clust, cent)
16:        newCentroids ← []
17:        for l ← 0, len(cvList) do
18:            if cvList[l] <= maxCulsterVar then
19:                newCentroids.append(cent[i])
20:            else
21:                newCentroids.append(splitCentInTwo(cent[i]))
22:            end if
23:        end for
24:        if len(cent) = len(newCentriods) then
25:            (finalClust, finalCent) ← (clust, cent)
26:            if minKFound = len(finalCent) then
27:                break
28:            end if
29:            if minKFound = −1 then
30:                minKFound ← len(finalCent)
31:            else
32:                minKFound ← min(k, len(finalCent))
33:            end if
34:        end if
35:        cent ← newCentroids
36:        k ← len(newCentroids)
37:    end for
38: end for
39: return finalClust, finalCent
```

Figure 4 shows the results of the clustering search method for Edison's jobs in 2014. This graph is a scatter plot of the jobs where each job is represented by a colored dot. The x-coordinate corresponds to the job's wall clock time and the y-coordinate to the number of cores allocated to the job. Note that the y-axis is in logarithmic scale. The execution queue of the job is identified by the color of the dot. The clusters' centroids are represented by black dots, while the color boxes are the boundary jobs observed in each cluster (minimum and maximum wall clock time and number of cores).

Table 5 shows the results of the clustering. Eight clusters were found for Carver, 11 for Edison and 12 for Hopper. This implies that Carver has a more homogeneous job set compared to Edison and Hopper As noted in the previous section, 70% of Carver jobs come from the *serial* queue,

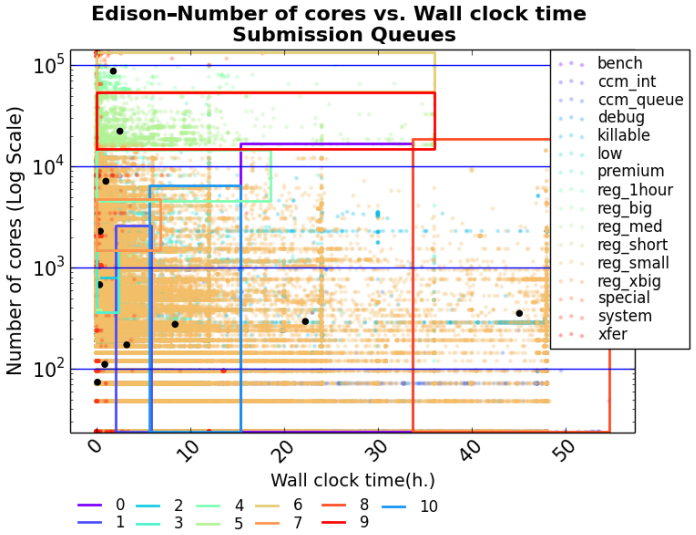


Figure 4: Result of the job clustering method for Edison 2014 with 8 clusters. Jobs are mapped on queues and clusters: Each dot is a job and dot color indicates the queue. Black dots are cluster centroids and color boxes are the surrounding jobs belonging to the same cluster. Clusters are sets of jobs with similar geometry.

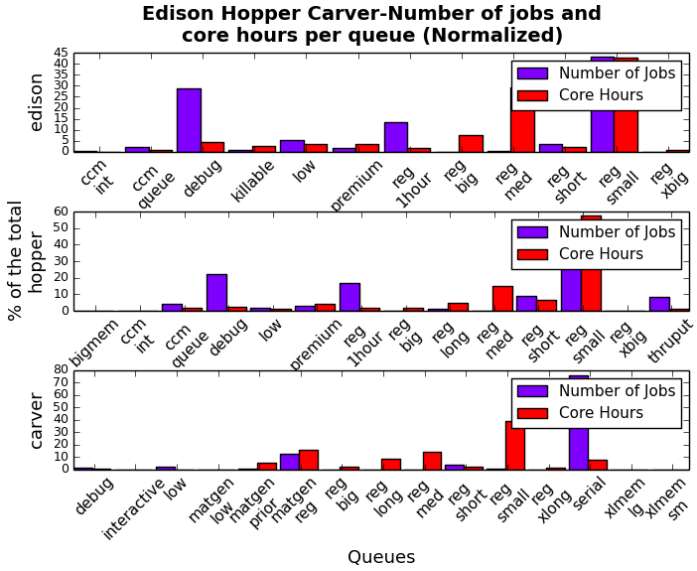


Figure 5: Normalized view of the number of jobs and core-hours per queue in Edison, Hopper, and Carver.

defined for single core jobs with long run times.

5. Queue Characterization

In the analyzed systems, job priorities are calculated depending on the queues that they are assigned to, also different job geometries were limited to be submitted to certain queues. As a consequence, queue configuration has an effect on how user submit jobs and how job priorities are distributed. In this section, we analyze the relationship between queue configuration, job geometry, and submission behavior.

5.1. Queue significance

The first step in analyzing the queue configuration and behavior is to understand the relevance of each queue in the system. For this, Figure 5 shows the normalized view of the number of jobs and core-hours per execution queue on Edison, Hopper, and Carver. The interpretation of this data is based on the configuration presented in Table 2.

On Hopper, the *debug* queue contains approximately 20% of the total jobs. The *debug* queue is typically used for testing and has a wall clock time limit of 30 minutes. The queue *reg_small* contains 30% of the jobs and approximately 56% core-hours. The queue *reg_med* presents a lower job count ($< 5\%$) and core-hours ($\sim 15\%$). The *thruput* queue admits 168 hours jobs and multiple submissions from the same user (600). However, its contribution to the total system utilization is less than 10% of jobs and 2% of core-hours.

Edison is very similar to Hopper, with the *reg_small* queue having $\sim 40\%$ of the jobs, and $\sim 40\%$ of the core-hours contributed by the *debug* and *reg_1h* queues.

Carver shows a different pattern. The *serial* queue contains more than 70% of the jobs, which consumed less than 10% of core hours. This percentage matches with the fact that this queue has exclusive access to 80 out of 1,180 total nodes ($\sim 7\%$) that are used for computation. The *reg_small* queue ($\sim 40\%$) and *matgen_reg* ($\sim 15\%$) account for the majority of the remaining core-hour usage.

5.2. Queue Diversity

The minimum number of clusters and their boundaries provide an overall view on the job diversity, and a possible recommendation for queue configuration. However, it gives no information about the job mix for a queue, i.e. how similar are the jobs in each queue. Thus, we define the *queue homogeneity index* as a new metric to compare the diversity of the jobs in a queue. After the clustering process, it is possible to identify each job's original cluster. Since jobs from different clusters are significantly different, a queue which jobs largely map on a single cluster will contain more homogeneous jobs than a queue whose jobs map to many clusters. The *queue homogeneity index* is the percentage of queue jobs that are mapped to the queue's dominant cluster, i.e. the cluster to which most jobs of the queue belong. For example, a queue maps jobs to three clusters with the following shares: 20%, 30%, and 50%. The cluster that contributes the most has 50% of the jobs and, as consequence, the *queue homogeneity index* is 50%. A higher index value indicates that many jobs are mapped to the same cluster and are thus geometrically similar to each other, while a lower index value means that the queue's jobs are more heterogeneous.

Table 5 present the queue homogeneity indices for the all the queues in Edison, Hopper, and Carver. This data allows to identify which queues contain more diverse job mixes and are candidate to be reviewed and, if needed, divided into smaller better defined queues. In Edison, reg_big, reg_med, and reg_big are the more homogeneous

Edison	11 c.	Hopper	12 c.	Carver	8 c.
Queue	/1	Queue	/1	Queue	/1
ccm_queue	0.46	bigmem	0.31	debug	0.32
debug	0.63	ccm_queue	0.45	interactive	0.35
killable	0.40	debug	0.70	low	0.99
low	0.53	interactive	0.85	matgen_low	0.62
premium	0.27	killable	0.45	matgen_prior	0.66
reg_1hour	0.71	low	0.59	matgen_reg	0.68
reg_big	0.96	premium	0.40	reg_big	0.70
reg_med	0.98	reg_1hour	0.69	reg_long	0.31
reg_short	0.42	reg_big	0.66	reg_med	0.82
reg_small	0.42	reg_long	0.50	reg_short	0.62
reg_xbig	1.00	reg_med	0.86	reg_small	0.26
		reg_short	0.39	reg_xlong	0.55
		reg_small	0.36	serial	0.87
		reg_xbig	1.00	usplanck	0.54
		thruput	0.77	xlmem_lg	0.24
				xlmem_sm	0.58

Table 4: Queue homogeneity indices for each machine: share of number of jobs belonging to its dominant cluster. In light green queues with indices in $(0.50, 0.75]$ interval, in darker green queues with indices $(0.75, 1.00]$.

queues while premium is the more diverse (0.27). In intermediate values (close to 0.50), reg_small appears to be diverse (0.42) and since its impact of the system is large (40% of jobs and core-hours), it is a good candidate of further study for division. Among Hopper's queues reg_small is quite diverse (0.36) and significant in the system (50% of jobs and core-hours), becoming a good candidate for revision. In Carver, reg_small (0.26) and significant (40% of core-hours) and should also reviewed.

Systems have different queue configurations and, in order to compare different systems with different workloads, a global metric is established. It aggregates the *queue homogeneity index* of all the queues by taking into account the queues' importance relative to the entire workload in the system.

Figure 5 presents two criteria for the impact of a queue on the system: number of jobs contained and amount of core-hours contributed. The former is useful to understand scheduler behavior, while the latter represents the fraction of the machine time the queue occupies. Missing one aspect of the system may give an incomplete picture, so we define two more metrics:

Job homogeneity index is calculated as a linear combination of the queue's homogeneity index. The coefficients are the share of jobs contained by the corresponding queue. For example, Queue1 has a homogeneity index of 0.6 and contributes 30% of the system's jobs. Queue2 has a homogeneity index of 0.4 and contributes 70% of the jobs. The *Job homogeneity index* is thus calculated as: $0.6 \cdot 0.3 + 0.4 \cdot 0.7 = 0.46$

Time homogeneity index is calculated as a linear combination of a queue's homogeneity, but in the time dimension. The coefficients are the shares of core-hours contributed to the system by the corresponding queue. In the example of Queue1 and Queue2: Queue1 contributes 30% of the system's jobs, and represents 80% of the core-hours of the system. Queue2 contributes 70% of the jobs that represent 20% of the system's core-hours. The *Time homogeneity index* is thus calculated as: $0.6 \cdot 0.8 + 0.4 \cdot 0.2 = 0.56$.

	Edison	Hopper	Carver
Clusters	11	12	8
Job homogeneity idx.	0.51	0.57	0.82
Time homogeneity idx.	0.64	0.49	0.51

Table 5: Queue analysis results: for each machine, minimum number of k-mean clusters discovered in the jobs and homogeneity indices.

We understand that larger indices imply that jobs in queues are more homogeneous, and policies are able to do more precise job prioritization for certain types of jobs. A lower index implies the existence of queues that contain a diverse job mix, which probably should be divided in more narrowly defined queues. The queue homogeneity index defined above can be used to determine what and how queues should be modified. These indices are not absolute measurements, and can only be used to compare similar systems (like the ones studied) or the same system during times.

Table 5 shows the calculated *homogeneity indices* for the three NERSC systems. The job homogeneity index shows that Carver is the system in which queues contain a reasonably uniform job mix. The time homogeneity index produces a different ordering: Edison, Carver, Hopper. This implies that the uniform queues on Carver have many jobs, but they are very small in terms of the total number of core-hours. On Hopper the uniform queues contains fewer jobs, but they contribute a significant part of the system's core-hours.

This analysis can be used to improve the job sorting across queues. From the point of view of core-hours, Edison has the best sorted queues. The queue *reg_small* is the largest contributor in terms of core-hours and jobs, however, only 42% of the queue jobs are mapped to the same cluster, implying a high diversity within the queue. The *reg_small* queue allows jobs up to 48 hours long and between 1 and 16,368 cores in size. Dividing this queue into subqueues with sub ranges of wall clock times or cores could have a positive impact on the time homogeneity index of Edison, and support improved job prioritization on the system.

6. Performance Characterization

In this section we study the performance of the systems from the perspectives of both resource providers (utilization) and users (wait time). This study includes a more detailed analysis on the jobs' wait time with a focus on its relationship with queue organization and job diversity.

6.1. Utilization

Facilities such as NERSC report the utilization of their resources periodically. The 2014 NERSC report calculates

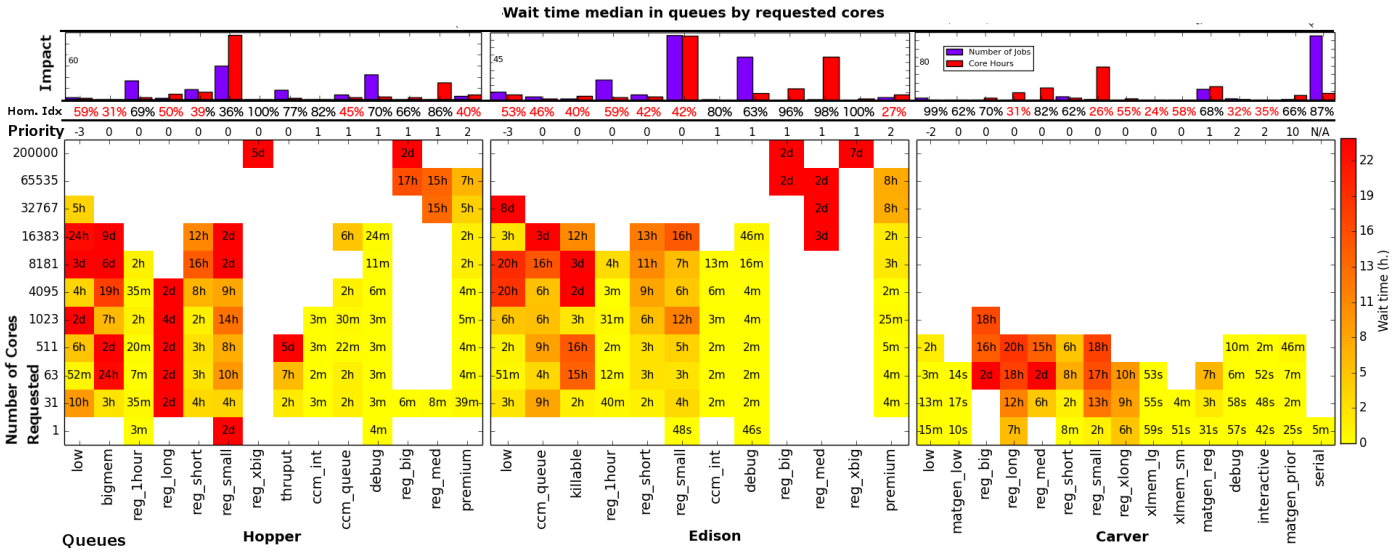


Figure 6: Job wait time median per queue depending on the requested cores. Aggregated on top: priority per queue, median of the wall clock time of queue jobs, jobs per queue (normalized), core-hours per queue (normalized).

System	Edison	Hopper	Carver
TSU	0.91	0.90	N/A
tTSU	0.87	0.80	0.88

Table 6: Total System Utilization (TSU) and theoretical Total System Utilization (tTSU). tTSU is under TSU since it does not take into account system maintenance down times.

the Total System Utilization (TSU) of Hopper and Edison as:

$$TSU = \frac{\text{core-hours used in period}}{\text{core-hours available in period}} \quad (1)$$

The available core-hours are calculated subtracting maintenance time (full and partial) and other temporal resource reductions. Down times are tracked manually and the TSU is calculated for reporting reasons. The available logs for this analysis do not contain system availability information. Thus, we calculate the *theoretical Total System Utilization* (tTSU) as:

$$tTSU = \frac{\text{core-hours used in period}}{\text{time period} * \text{maximum system capacity}} \quad (2)$$

By definition, tTSU will be less than or equal to TSU. We present the reported TSU and the tTSU in Table 6. Carver's TSU was not available for 2014.

6.2. Job wait time

Previous analysis (Section 4.2) presents a coarse grained analysis of jobs' wait time. To understand the performance of the system, it is important to also understand wait times relative to job geometry and queue priorities. In this section, we detail our wait time distribution analysis, i.e. we calculate the median wait time of jobs grouped by queues (and thus corresponding priority) and job geometry. User wall clock time estimations of the systems are inaccurate (Section 4.2), as a consequence we use the number of cores requested as job geometry metric for this analysis.

We present job wait time for the three systems as heat maps in Figure 6. For each system, queues appear on the x-axis, ordered by priority. The y-axis represents a non-linear categorization of the possible numbers of cores allocated to jobs. Each square contains the wait time median (in seconds, minutes, hours, or days) for a particular queue that was allocated the cores specified on the y-axis. White regions indicate that there were no jobs with the specific queue and cores combination. A darker tone or red represents longer wait times (24 hours or longer), and a lighter tone or yellow represents shorter wait times. The priority of each queue is specified above the heat map and below the bar graph. The bar chart on top shows the number of jobs and core-hours contributed by each queue to each system.

In all systems, we observe that the graph is darker at the top left corner and lighter at the bottom right. Thus, jobs in the same queue with a larger degree of parallelism have longer wait times. The effect follows from the fact that the wider jobs are harder to fit during scheduling. Also, jobs with similar number of cores have shorter wait times in queues with higher priorities. While these capture the general trend, we look more closely at the anomalies in the heat map.

On Hopper, the *low* queue has the lowest priority (-3) and longer wait times than most of the other queues. The queues with priority zero, *reg_1h*, *reg_xbig*, and *reg_short*, present the expected behavior: longer wait times than the ones with priority one and shorter than the one with -3. The *bigmem*, *reg_long*, *reg_small*, and *thruput* queues, present wait times significantly higher. The *big_mem* queue is the gateway for the nodes with more memory (384 large memory nodes vs. 6,000 regular nodes). The jobs in this queue may be experiencing higher wait times because they

compete to use a smaller resource set. The *reg_long* and *thruput* queues contain longer jobs than the rest with the same priority (also much longer than the ones in *low*), and thus might not be able to take advantage of backfilling. Finally, the *reg_small* long wait times may be related to its large contribution of jobs and core-hours. The queues with higher priorities than zero show shorter wait times.

Wait time for jobs allocating different number of cores but in the same queue presented unexpected values (i.e. longer wait times for larger number of cores allocated). In some cases it could be related to the jobs' wall clock time, as is the case for the *big_mem* queue. Its wait time for the 64 to 511 cores range is two days, while between 512 to 1023 cores is seven hours. We analyzed the median of wall clock times in those ranges, obtaining one day and three hours respectively. The jobs allocating 64 to 511 cores were probably harder to backfill due to their longer wall clock times, increasing the wait time.

Edison exhibits a similar behavior to Hopper. The queues *killable* and *ccm_queue* have longer wait times, because their job's run-time is longer than the jobs in other queues with similar or lower priority. The *reg_small* queue has the maximum jobs and core-hours used on Edison, resulting in possibly longer wait times of its jobs. The jobs with higher priorities behave as expected, showing shorter wait times.

Carver displays different trends compared to Hopper and Edison. The *serial* queue has exclusive resources and a median wait time of five minutes. The *matgen* queues has a pool of resources, but those might be used by other queues. The queue also has a high job count, and thus higher wait times than queues with lower priority. The *xlmem* queues are similar to Hopper's *bigmem*, and meant to serve jobs with large memory requirements. However, their resource assignment is different: On Hopper, *bigmem* jobs can only be executed on nodes with large memory capacity, but these nodes can also execute jobs from other queues. In the case of Carver, only the jobs from the *xlmem* queues can be run in the special nodes. This exclusive access, combined with *xlmem*'s low job count and core-hour contribution, explains why these queues' median job wait time is under four minutes.

Three queues (*reg_big*, *reg_long*, reg_xlong) have priority zero and longer wait times than the other queues. The *reg_small* queue jobs consume more core-hours than any other queue (apart from *serial*), which may be the reason for the long wait times in this queue. Finally, the queues with higher priorities behave as expected.

In general we observed that queues that did not display expected queue wait time patterns had low homogeneity indices (under 60%). In particular, the queues on Hopper with such indexes should be further studied. More predictable wait times could be possible by dividing these queues according to the observed job clusters.

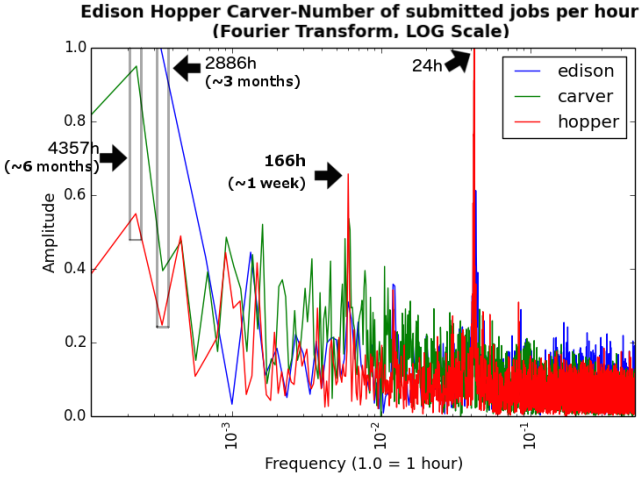


Figure 7: Fourier decomposition of the series of the jobs submitted per hour for Edison, Hopper, and Carver to detect dominant submission cycle. Note the logarithmic scale for the frequencies. Most powerful frequencies highlighted with a black arrow and its corresponding period.

7. Trend Analysis

After analyzing the system behavior during a particular period of time, this section presents a similar analysis but performed over the lifetime (Jan 2010 to June 2014) for Hopper and Carver. The purpose is to observe if there is any clear evolution pattern.

7.1. Time Patterns and analysis granularity

Choosing a time period to slice the data was the first step of this analysis. A Fourier transform analysis was performed on the number of tasks submitted per hour, since this analysis allows us to detect cycles in the user behavior [22]. The result can be observed in Figure 7: Black arrows point to the most powerful frequencies correspond to the periods of 1 day, 1 week, 3 months, 6 months. This data matches human period times (days, working weeks). Each project has a number of core-hours to be used in a year, divided in 4 allocation quarters in which the project has to consume (or forfeit) the corresponding allocated time. The strong pattern around the allocation year led us to choose one year as the trend analysis period time.

7.2. Job geometry

The evolution of the first job geometry variable is presented in Figure 8 as a box plot of job wall clock time for each system in each year. Hopper shows a significantly low wall clock time median in 2010 ($<$ 1 minute), which might be related to the fact that it was a smaller testbed system that year. In 2011, the median increased to $\sim$ 5 minutes and subsequently increased to $\sim$ 12 minutes by 2014. Carver shows a different trend: the median, upper and lower quartile decrease effectively over the period studied. The median decreased from $\sim$ 20 minutes (2010) to $\sim$ 6 minutes (2014). However, there is some variation

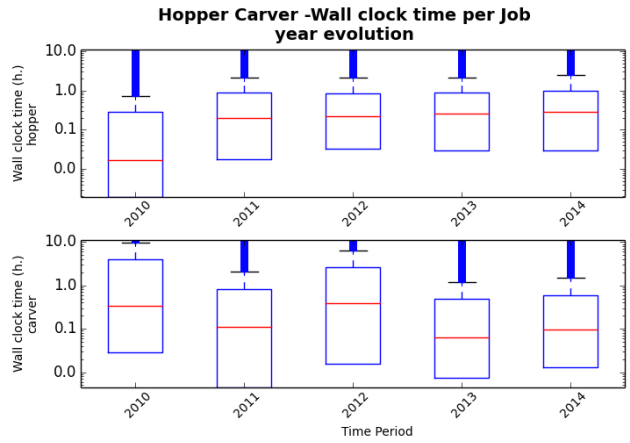


Figure 8: Job wall clock time for each each workload year. Trend: Hopper jobs become longer, Carver jobs shorter. Majority of jobs under one hour.

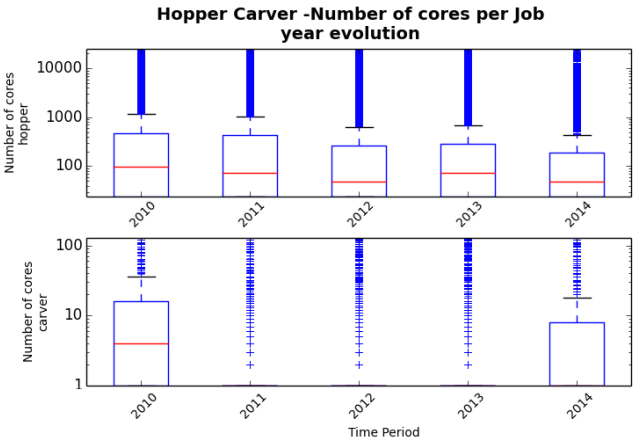


Figure 9: Allocated number of cores for each workload year. Trend: Hopper jobs allocate less cores. In 2011-2013, most Carver jobs used one core.

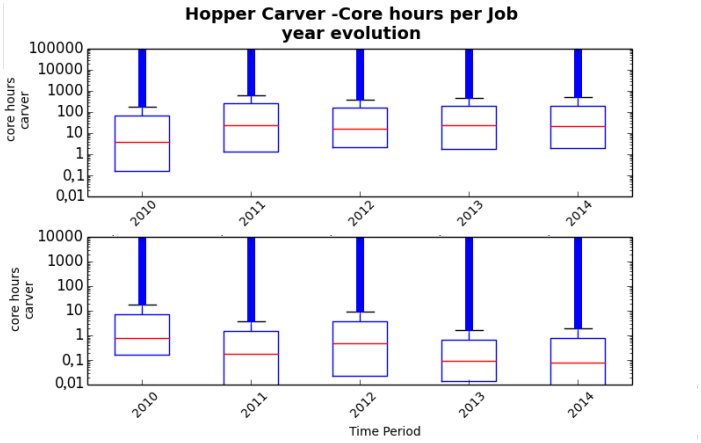


Figure 10: Allocated core-hours for each workload year. Trend: No changes on Hopper. Carver jobs become smaller.

Figure 11: Jobs' wait time evolution for each workload year. Trend: All systems increase wait time. Carver lower wait time in 2011.

from year to year: It is observed that in the first year in production, Carver ran longer jobs than Hopper, a fact that slowly changed in 2014 when Hopper ran longer jobs that Carver. More generally, Hopper presents fairly short jobs, as the highest upper quartile is around the one hour value. Carver presents a similar behavior as the upper quartiles of the last years are under one hour.

The evolution of the width of jobs (number of allocated cores per job) is shown in Figure 10. For Hopper, the median decreases from 100 cores (2010) to under 30 cores (2014). Carver presents an opposite pattern. Except for 2010, the median of the rest of the years is one core, showing the predominance of single core serial jobs. In 2014, the upper quartile increased to 8 cores.

The core-time i.e., total clock time across all cores was also studied. In the case of Hopper, it remains nearly the same through time with a median of $\sim$ 20 core-hours and the upper quartile slightly under 200 core-hours in most years. In the case of Carver, it slowly decreases from a median of almost 1 core-hour to $\sim$ 6 core minutes (and a last upper quartile of 1 core-hour).

In summary, Hopper jobs (shorter jobs, with a higher degree of parallelism, bigger than Carver's) seem to be showing an increase in their wall clock time. As the effective job's core-hours remain the same, they must be using fewer cores. Carver jobs (longer jobs, lower degree of parallelism, fewer core-hours than Hopper's) have decreasing wall clock time and use more cores, but the increase is not sufficient to keep the job's core hours steady over the years.

7.3. Job wait time

According to Figure 11, for Hopper, the median of the wait time is steadily increasing from under 100 seconds to over 20 minutes (a pattern also present in the upper and lower quartiles). On Carver, the effective wait time increases over the four years from $\sim$ 10 minutes in 2010 to $\sim$ 20 minutes in 2014. However we notice a zigzag pattern trend in between. In 2011, Carver presented significantly shorter wait times, which could be attributed to a known increase of resources in the system. The steady increase of

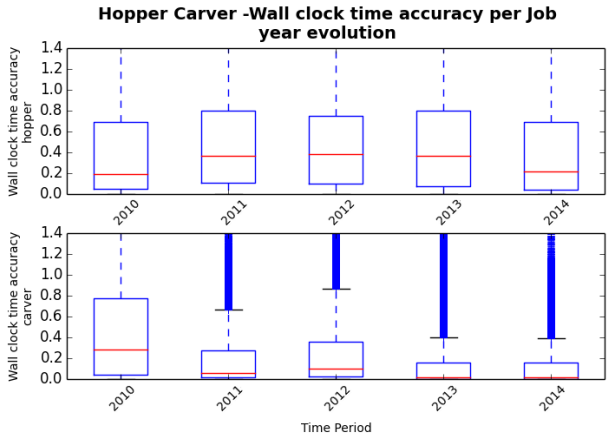


Figure 12: Jobs' wall clock time accuracy evolution for each workload year. In all systems wall clock time remains low.

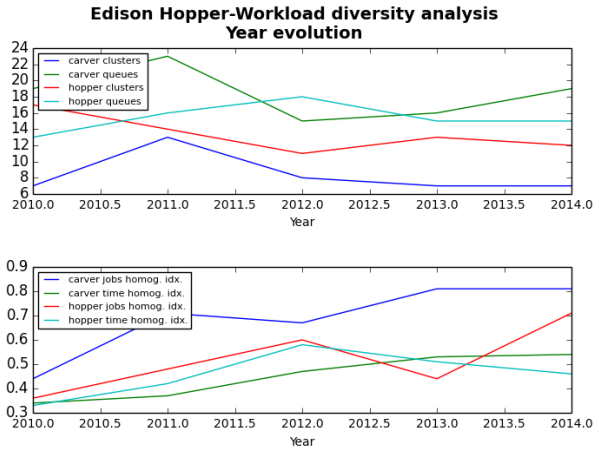


Figure 13: Workload diversity and in queue homogeneity index: Overall workload becoming less diverse. Job mix in queues becoming more uniform.

wait time over the lifetime fits with the growth of the user community of the systems.

7.4. Wall clock time accuracy

The wall clock time accuracy is calculated as *real/estimated* wall clock time. The results are shown in Figure 12. Hopper does not show a clear trend: 2011 to 2013 presents a higher accuracy than 2010 and 2014, with a median variation between 0.2 and 0.4. For Carver, the median decreases over time, with significant changes between 2010 (~ 0.25) and 2011 (<0.1). In 2014, the median is under 0.1 and the last quartile it is under 0.2. For Carver, there is a clear pattern of worse estimations as the time proceeds. In general, both systems present very low values with medians under 0.4.

Wall clock accuracy time does not show a noticeable pattern beyond the fact that accuracy is low. On Hopper, 50% of all jobs run less than 40% of the estimated time. Similarly on Carver, 50% of all jobs less than 20% of the estimated time. These values indicate that the decisions made by the backfilling algorithms are based on inaccurate user estimations.

7.5. Job and queue diversity

Following the methodology presented in Section 3.3, the diversity analysis was performed for each year and system under two different perspectives: how different are the jobs overall and how different are the jobs inside of each queue. Results can be observed in Figure 13.

Hopper shows lower numbers of clusters over time, decreasing from 17 to 12 clusters, implying a decreasing general diversity. In the case of Carver, it started with a fairly simple job mix (7 clusters), had an increase of complexity in the second year (13 clusters), to go down to the same number of clusters (7) in the last two years. In all years Carver presents a more homogeneous job mix in comparison to Hopper.

According to Figure 13, Hopper's homogeneity *job homogeneity index* increased 0.36 to 0.71, while its *time homogeneity index* increased from 0.33 to 0.46. Queues with more jobs contain more self similar jobs in 2014 than in 2010, It is interesting to note that 2012 was higher than in other years, implying that queues contributing more core-hours had more self similar jobs. In the case of carver the trend is similar, *job homogeneity index* increases from 0.44 (2010) to 0.81 (2014) and *time homogeneity index* increases from 0.34 (2010) to 0.54 (2011). As we observe all years, it can be stated that under both criteria Carver queues job diversity was lower than in Hopper.

Overall, it can be concluded that, as both systems aged, the jobs submitted by the users evolved to become more uniform. Additionally, the configured policies have evolved to build queues that classify better the jobs to contain more self similar jobs.

8. Results summary and conclusions

In this section, we summarize our results and present our conclusions on the workload analysis method and results characterization.

8.1. Summary of results, a year of workload

We present a reference of the current state of the workloads of two large scale and one mid scale high performance systems: We summarize the key results from our detailed analysis performed on the 2014's workload from Edison, Hopper, and Carver. We also compare them with pre-existing analysis of similar, still older.

- The job wall clock times are short on all three systems. From 86% to 88% of the jobs run less than 2 hours.

- On Edison and Hopper, 37%, 39% of the number of jobs run on one node and 69%, 75% run on 10 nodes or less. On Carver, 92% of the jobs run on one node.
- On Carver, 77% of its jobs require one or less core-hours. Carver jobs use far fewer hours than jobs on Hopper and Edison.
- Jobs run less than 50% of their requested time: 60% of Edison jobs, 66% of Hopper's jobs and 95% of Carver jobs. Jobs run over their underestimated wall clock time: 10% Hopper's jobs, 11% of Edison's and 8% of Carver's.
- Carver has the most homogeneous workload (more similar jobs, similarity among jobs in the same queue). Hopper has a diverse workload with a complex job mix in its queues.
- Carver has the longest wait times, although its jobs allocate significantly fewer core-hours than jobs on Hopper and Edison.
- On all systems, the wait time increases as jobs allocate more resources. The wait time decreases with higher priority in most cases. In some cases anomalies are observed; e.g. larger jobs with lower priority experience shorter wait time.

Although these results represent the state of current systems, it is relevant to understand their difference to exiting work on workloads analysis of similar systems. In particular, our results were compared to analyses on Intrepid an Stampede, one current and one past HPC systems.

Intrepid was a Blue Gene/P supercomputer, with 163,840 cores, 80 TB of memory (512 MB per core), custom interconnect, peak Linpack performance of 458.6 TFLOPS, and was deployed in 2008 at the Argonne National Laboratory. From the point of view of configuration, Intrepid is more similar to Carver, as both are Teraflop systems and closer in deployment time. However, Intrepid is a Blue Gene/P system, characterized by providing compute power through smaller but more numerous CPU cores, an opposing approach to the analyzed NERSC systems' architecture, based on more powerful cores. We compare with a trace from nine months of Intrepid in 2009 [1].

Stampede is a POWEREDGE C8220 high performance cluster with 462,462 cores, an Infiniband interconnect, that can deliver up to 8 PFLOPS. It was deployed at the Texas Advanced Computing Center (Univ. of Texas) in 2012. Stampede could be compared to Edison or Hopper in terms of capacity, but its architecture differ from them as its processing units are hybrid. Stampede includes both Xeon and Phi processors in its compute nodes. For applications using its Xeon processors, Stampede performs like Edison or Hopper, in fact, Edison's processor are the next generation (Ivy Bridge) to Stampede's (Sandy Bridge). However, the Phi processors are different. They are manycore processors that include 61 light but power efficient CPU cores, from a generation before current Knightslanding (KNL) manycore Intel chips [25]. Phi processors require a regular processor to load work on them and manage their operations. We compare with an analysis over a trace of three months of Stampede in 2013 [26].

Looking into the workloads, Edison and Hopper's wall clock time distribution matches the patterns observed on Intrepid [1]. Carver's run time CDF is steeper and similar to Stampede [26]. Jobs in all three systems use fewer cores than Intrepid. Edison and Hopper jobs are similar to the jobs on Stampede in terms of cores.

The *serial* queue jobs on Carver dominate the distribution. Thus, Carver jobs are very different from Edison and Hopper and other HPC systems. Edison and Hopper share characteristics with reference systems like Intrepid or Stampede. It is possible that current DOE Leadership Computing Facilities exhibit slightly different workload characteristics [27]. This work is a first step to understand if results from NERSC may translate to such facilities.

8.2. Summary of results, systems lifetime evolution

Observing the evolution of large scientific infrastructures through their life time allows to understand systems evolution as their use mature. It also provides data to support the characteristics of future workloads.

Figure 13 presents how Hopper's workload evolved to a more uniform job mix. However, Carver presents a spike in the second year of its lifetime, to return to values similar to the beginning. Hopper results capture the nature evolution of systems, where the scheduler and other machine characteristics are refined based on the workload characteristics. Carver suffered two significant changes in 2011 that might have affected its diversity. First, Carver was expanded in 2011 [28]. Second, the serial batch queue (long running, low degree of parallelization) was added [29].

Figure 11 presents how the wait time steadily increases as the systems age, this fits with a growing scientific community using the same system and more advanced applications that require faster infrastructure. Still, there is one exception, in 2011 Carver presented significantly smaller wait times. This could be attributed to its expansion in 2011.

As Hopper and Carver are compared in Figure 8 and Figure 10, the analysis on the evolution of the job geometry reveal that Hopper jobs (which had shorter jobs but with a higher degree of parallelism than Carver) seems be increasing their wall clock time but using fewer CPU cores, while Carver jobs (which had longer jobs but with a lower degree of parallelism than Hopper) are decreasing their wall clock time and using more CPU cores.

In general, it is interesting to observe signs that would support the idea that as a systems ages, its workload variables evolve with a distinct trend, becoming more homogeneous but putting an increasing pressure on the available resources.

System	Vendor	Model	Built	Nodes	Cores/N	Cores	Memory	Network	TFlops/s	Processor
Intrepid	IBM	Blue Gene/P	2008	40,960	4	163,840	80 TB	Torus	557.1	Blue Gene
Stampede	Dell	PowerEdge	2012	6,400	16+61	462,462	192 TB	Inf. FDR	8,520	Xeon, Phi

Table 7: Intrepid and Stampede characteristics

8.3. Conclusions

Analyzing NERSC's workload offered a challenge that required new analysis tools. For this, we establish a methodology that include traditional workload analysis techniques (e.g., CDF analysis of job variables) but incorporates new methods to asses job heterogeneity. The job heterogeneity analysis includes a novel algorithm that employ's k-means clustering to detect the minimum number of dominant job geometries in an HPC workload. The method also analyzes the mapping of dominant job groups on the system prioritization schema and the resulting job wait times. This enables to asses the effect of job heterogeneity on the the scheduling performance in terms of wait time.

The results of the first application of this methodology establish a reference of the state of the workload in 2014 of three high performance systems (Edison, Hopper, and Carver). Such systems are similar size, architecture, and workload to many other current HPC systems. These results can be of use to understand the behavior in other systems, and among them we highlight: (1) The job geometries were fairly diverse including significant number of smaller jobs compared to older systems. The low per queue homogeneity indexes, show that (2) single priority policies are affecting jobs with a fairly diverse geometry. The wait time analysis shows that (3) studied queues with low homogeneity indexes present poor correlation between job's wait time and geometry. Job's submission patterns show that (4) the accuracy of users' predictions of their job's wall clock time (fundamental for the performance of backfilling functions) is very low, and does not improve over time. Finally, (5) Hopper and Carver workloads presented a clear trend in their four year lifetime: they become less diverse, their queues classify better their jobs, and they become more similar. (6) Also, they experience a heavy load that increases the overall wait times.

Our results and methodology are of use for future scheduling research and systems operations management. Scheduling research needs to address present and future workloads and our results set a first step to understand characteristics of future systems (e.g., diverse jobs, smaller jobs, or low accuracy in runtime estimations).

For system management, we highlight a result and an alternative application of our methodology. First, low values on wall clock time accuracy points to further research on how to encourage users to encourage users to provide better predictions. Better runtime accuracy will increase the quality of the backfilling in schedulers. Finally, the dominant job groups produced by the job heterogeneity analysis could be a template to define priority groups and queues. Our results show that diverse queues offered hard to predict wait times. Queues obtained by subdividing dominant job groups could show predictable wait times.

9. Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) and the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Financial support has been provided in part by the Swedish Government's strategic effort eSSENCE, by the European Union's Seventh Framework Programme under grant agreement 610711 (CACTOS), the European Unions Framework Programme Horizon 2020 under grant agreement 732667 (RECAP), and the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control. We would like to thank Sophia Pasadis for editing help with the paper.

References

[1] D. Feitelson, Parallel workloads archive 71 (86) (2007) 337–360, `http://www.cs.huji.ac.il/labs/parallel/workload`.

[2] NERSC, `http://www.nersc.gov`, 2015-01-18.

[3] G. P. R. Alvarez, P.-O. Östberg, E. Elmroth, K. Antypas, R. Gerber, L. Ramakrishnan, Towards understanding job heterogeneity in hpc: A nersc case study, in: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2016, pp. 521–526.

[4] G. Rodrigo, P.-O. Östberg, E. Elmroth, K. Antypass, R. Gerber, L. Ramakrishnan, HPC system lifetime story: Workload characterization and evolutionary analyses on NERSC systems, in: The 24th International ACM Symposium on High-Performance Distributed Computing (HPDC), 2015.

[5] M. A. Bauer, A. Biem, S. McIntyre, N. Tamura, Y. Xie, High-performance parallel and stream processing of x-ray microdiffraction data on multicores, in: Journal of Physics: Conference Series, Vol. 341, IOP Publishing, 2012, p. 012025.

[6] S. N. Srirama, P. Jakovits, E. Vainikko, Adapting scientific computing problems to clouds using mapreduce, Future Generation Computer Systems 28 (1) (2012) 184–192.

[7] T. Hey, S. Tansley, K. M. Tolle, et al., The fourth paradigm: data-intensive scientific discovery, Vol. 1, Microsoft research Redmond, WA, 2009.

[8] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, Parallel job scheduling, a status report, in: Job Scheduling Strategies for Parallel Processing, Springer, 2005, pp. 1–16.

[9] D. A. Lifka, The ANL/IBM SP scheduling system, in: Job Scheduling Strategies for Parallel Processing, Springer, 1995, pp. 295–303.

[10] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. Epema, The grid workloads archive, Future Generation Computer Systems 24 (7) (2008) 672–686.

[11] A. K. Mishra, J. L. Hellerstein, W. Cirne, C. R. Das, Towards characterizing cloud backend workloads: insights from google

compute clusters, ACM SIGMETRICS Performance Evaluation Review 37 (4) (2010) 34–41.
[12] K. Antypas, B. A. Austin, T. L. Butler, R. A. Gerber, NERSC workload analysis on Hopper, Tech. rep., LBNL Report: 6804E (October 2014).
[13] NERSC, Submitting batch jobs (carver), https://www.nersc.gov/users/computational-systems/carver/running-jobs/batch-jobs/, 2015.1.15.
[14] C. Vaughan, M. Rajan, R. Barrett, D. Doerfler, K. Pedretti, Investigating the impact of the Cielo Cray XE6 architecture on scientific application codes, in: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), IEEE, 2011, pp. 1831–1837.
[15] T. M. Declerck, I. Sakrejda, External Torque/Moab on an XC30 and Fairshare, Tech. rep., NERSC, Lawrence Berkeley National Lab (2013).
[16] Y. Etsion, D. Tsafrir, A short survey of commercial cluster batch schedulers, School of Computer Science and Engineering, The Hebrew University of Jerusalem 44221 (2005) 2005–13.
[17] G. Staples, Torque resource manager, in: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, ACM, 2006, p. 8.
[18] K. Antypas, NERSC-6 workload analysis and benchmark selection process, Lawrence Berkeley National Laboratory.
[19] NERSC, Queues and polices (carver), https://www.nersc.gov/users/computational-systems/carver/running-jobs/queues-and-policies/, 2014.1.15.
URL https://www.nersc.gov/users/computational-systems/carver/running-jobs/queues-and-policies/
[20] J. Weinberg, A. Snavely, Symbiotic space-sharing on sdsc's datastar system, in: Job Scheduling Strategies for Parallel Processing, Springer, 2007, pp. 192–209.
[21] J. D. Hunter, Matplotlib: A 2D graphics environment, Computing In Science & Engineering 9 (3) (2007) 90–95.
[22] W. W.-S. Wei, Time series analysis, Addison-Wesley publ, 1994.
[23] A. Coates, A. Y. Ng, Learning feature representations with k-means, in: Neural networks: Tricks of the trade, Springer, 2012, pp. 561–580.
[24] J. A. Hartigan, M. A. Wong, Algorithm as 136: A k-means clustering algorithm, Applied statistics (1979) 100–108.
[25] A. Sodani, Knights landing (knl): 2nd generation intel® xeon phi processor, in: Hot Chips 27 Symposium (HCS), 2015 IEEE, IEEE, 2015, pp. 1–24.
[26] J. Emeras, Workload traces analysis and replay in large scale distributed systems, Ph.D. thesis, Grenoble INP (2014).
[27] S. Ahern, S. R. Alam, M. R. Fahey, R. J. Hartman-Baker, R. F. Barrett, R. A. Kendall, D. B. Kothe, R. T. Mills, R. Sankaran, A. N. Tharrington, et al., Scientific application requirements for leadership computing at the exascale, Tech. rep., Oak Ridge National Laboratory (ORNL); Center for Computational Sciences (2007).
[28] NERSC, Magellan batch queues on carver, http://www.nersc.gov/REST/announcements/message_text.php?id=1991, 2015.01.15.
URL http://www.nersc.gov/REST/announcements/message_text.php?id=1991
[29] NERSC, Serial queue on carver/magellan, http://www.nersc.gov/REST/announcements/message_text.php?id=2007, 2015.01.15.
URL http://www.nersc.gov/REST/announcements/message_text.php?id=2007