Title
Towards understanding HPC users and systems: A NERSC case study

Authors
Rodrigo, Gonzalo P
Östberg, P-O
Elmroth, Erik
et al.

Publication Date
2018

DOI
10.1016/j.jpdc.2017.09.002

Peer reviewed

J. Parallel Distrib. Comput. 111 (2018) 206–221

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Towards understanding HPC users and systems: A NERSC case study

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

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

HIGHLIGHTS

- A method to perform analysis of HPC systems' workloads is proposed including per year detailed and time evolution analyses.
- A method to measure heterogeneity in job geometry is proposed.
- State of workload of three reference HPC systems are presented.
- Job geometry heterogeneity in queue is shown to affect wait time predictability.

ARTICLE INFO

Article history:
Received 20 April 2017
Received in revised form 18 August 2017
Accepted 4 September 2017
Available online 14 September 2017

Keywords:
Workload analysis
Supercomputer
HPC
Scheduling
NERSC
Heterogeneity
k-means

ABSTRACT

High performance computing (HPC) scheduling landscape currently faces new challenges due to the changes in the workload. Previously, HPC centers were dominated by tightly coupled MPI jobs. HPC workloads increasingly include high-throughput, data-intensive, and stream-processing applications. As a consequence, workloads are becoming more diverse at both application and job levels, posing new challenges to classical HPC schedulers. There is a need to understand the current HPC workloads and their evolution to facilitate informed future scheduling research and enable efficient scheduling in future HPC systems.

In this paper, we present a methodology to characterize workloads and assess their heterogeneity, at a particular time period and its evolution 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 (2010–2014).

1. Introduction

High performance computing (HPC) supports scientific research by providing large-scale resources to run simulations. Such applications are composed as tightly coupled MPI models, that have dominated HPC workloads. However, the workload characteristics on HPC systems has evolved in the last few years. 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, advances in experimental devices enable real experiments to produce higher resolution data that requires real-time processing. These trends have resulted in workloads becoming more diverse necessitating the need to support high-throughput, data-intensive, and stream-processing applications. It is unclear if current HPC schedulers produce optimal decisions for these workloads, since their focus has been on MPI dominated applications.

The evolving diverse workload requires that new scheduling models are 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. Workload characterization efforts are necessary to inform both short-term and long-term scheduling decisions. For example, workload characterization has informed the design of a real-time job queue on Cori [20], the most recent petascale system at the National Energy Research Scientific Computing Center (NERSC). The characterization showed

* Corresponding author at: Lawrence Berkeley National Lab Berkeley, CA 94720, USA.
E-mail addresses: gprodrigoalvarez@lbl.gov (G.P. Rodrigo), p-o@cs.umu.se (P.-O. Östberg), elmroth@cs.umu.se (E. Elmroth), kantypas@lbl.gov (K. Antypas), ragerber@lbl.gov (R. Gerber), lramakrishnan@lbl.gov (L. Ramakrishnan).

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

http://dx.doi.org/10.1016/j.jpdc.2017.09.002
0743-7315/

that the short run time of debug jobs guarantees frequent liberation of resources. As a consequence, any top priority job submitted to the *debug* partition will start within two minutes of its submission.

Previous efforts on workload modeling [11], while useful, are not representative of the top recent HPC systems. Also, previous work did not focus on the system queue configuration or 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.

In this paper, we combine and extend previous work on workload analysis [2,31]. First, we expand on the methodology to analyze system workloads in detail [2] by including a distribution analysis on job variables (degree of parallelism, run time, core-hours, inter-arrival time, and run time estimation accuracy) and a study on the system utilization estimation. This completes the characterization of the three systems with all the job related information needed to inform future job-scheduling research. We also detail a method to study the job geometry (allocated resources and run time) diversity. This method employs *k*-means clustering to identify dominating groups of jobs in the workload according to their geometry (run time and allocated CPU cores). In this work, we include also the algorithms needed to calculate the minimum number of job clusters. The extensions presented in this paper are critical for reproducibility and enables other researchers to apply these methods. Also, we perform an analysis of the correlation of queue diversity and wait times for jobs of different sizes. This analysis attempts to understand if the wait time of jobs matches the expected values according to priority and system configuration (i.e., larger jobs should wait longer, higher priority jobs should wait shorter) or if they deviate due to job heterogeneity. As an addition to previous work, we add *requested run time* as another variable in the analysis. This allows us to perform a complete analysis on wait times depending on all job-related variables that affect it including requested CPU cores, run time, job priority, and associated resource set size.

We also extend a second piece of work that characterizes HPC systems through their life by analyzing its workload variables evolution [31]. In this work we add the analysis of geometry homogeneity evolution. This allows understanding if the changes on the application landscape run on HPC system have an impact on the geometry of the jobs.

We apply our complete evaluation methodologies to the workloads of three systems (Hopper, Carver, and Edison) at the National Energy Research Scientific Computing Center (NERSC) [26] that are representative of current HPC systems. 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 recent 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. Our results establish the foundation necessary to predict future HPC workloads for designing future resource management models.

Our workload analysis methodology can also be used to inform 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. run time 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, our contributions in this paper include:

- We present a full description of the methods and algorithms required to perform workload diversity analysis and measure the self similarity of the jobs in the workload and their mapping on to the prioritization queues.
- We define a method to analyze the wait time of jobs depending on their geometry, queue priority, and diversity. We extend our previous work that did not take into account the run time of jobs in the analysis.
- We provide a detailed job, queue, performance, and diversity characterization of the NERSC workload, and their evolution over time. The results enable us to understand the users, system behavior, and the effect of queue heterogeneity on job 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 our conclusions in Section 9.

2. Background

In this section, we describe the challenges in the HPC community that motivate this work, introduce basic concepts on parallel job scheduling, and review previous work on 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 scheduling models and system architectures. In this section, we highlight some workload 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. However, advance reservations 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).

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 [16,33]. 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 at HPC centers. 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. 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

Table 1
Edison, Hopper, and Carver characteristics.

System	Vendor	Model	Built	Nodes	Cores/N	Cores	Memory	Network	TFlops/s	Service
Hopper	Cray	XE6	2010	6384	24	154,216	212 TB	Gemini	1280	Jan'10
Edison	Cray	XC30	2013	5576	24	133,824	357 TB	Aries	2570	Jan'13
Carver	IBM	iDataPlex	2010	1120	8/12/32	9,984	147 TB	Infiniband	106.5	Apr'10

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. Understanding the impact of the application diversity on the system at its most basic level – job geometry – motivates our workload heterogeneity analysis.

2.2. Scheduling

HPC schedulers optimize job placement to achieve the highest system utilization possible with a reasonable turn-around time according to the job priority. The most common base technique in schedulers is FCFS (First-Come, First-Served) [13]. In 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 fill resource gaps produced by 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 [23].

The quality of the results of the backfilling algorithm depends on the user's wall clock time estimation [13]. 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 job 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

Comparable traces to the ones analyzed in this work can be found in the Grid Workloads Archive [19] and the Parallel Workload Archive [11]. The archives contain job and performance characteristics (run time, parallelism, inter-arrival time, wait time, disk space, and memory). However, analyzed systems are either at least 10 years old or significantly smaller than the current top HPC systems.

Some supercomputing centers provide reports that complement this work by including insights on the workload applications. The Blue Waters workloads report [21] describes the workload, applications, and performance of a petascale university super computer. Workload analysis of Mira [32], a Blue Gene system at Argonne Leadership Computing Facility, provides insight on the user behavior in an HPC meant to support the scientific problems that require extreme parallelism. A previous analysis [4] 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 job geometry and memory requirements. This study throws the first insights towards application heterogeneity: e.g., Lattice Quantum Chromodynamics (QCD) applications that support moldable jobs (i.e., geometry can be decided at submission time), consumed more than 25% of 2012's Hopper core-hours.

Our work extends previous analyses and reports by addressing job heterogeneity and performing a comparative analyses of three recent systems of different sizes and characteristics: Edison was deployed in 2014 and still ranks 72 in the Top 500 list in June 2017, Hopper was a petascale system that was retired in late 2015, and Carver a commodity-hardware based terascale cluster retired in 2015.

Workload analysis methodologies used in our work are based on methodologies from Feitelson's book on workload analysis [12]. We model job variables as empirical distributions [24] and present them as cumulative distribution functions in Figs. 1 and 2. For the trend analysis in Section 7, patterns periods were detected by performing a Fourier spectral analysis [17] of the inter-arrival time of the jobs. However, in this work, user behavior is not modeled [40] because such analysis is only adequate to recreate workload traces which depend on the behavior of the scheduler [14] that process them.

As an important factor for scheduling performance, there is extensive work on user's run time estimation accuracy. For instance, users' accuracy is modeled [35] to support jobs scheduling research. Also, some research job schedulers use automatic run time predictions, instead of user provided. [36]. Finally, in previous work, it has been observed that users do better estimations under circumstances in which jobs are not killed if they run longer than requested [22].

Existing work on HPC workload analysis does not address job heterogeneity. Instead, our workload diversity analysis is inspired by methods used for industrial workloads [25]. This methodology introduces *k*-means as a tool for job similarity clustering and is extended in this work to include per queue analysis and a new methodology to compare the degree of heterogeneity across systems and system states.

3. Methodology

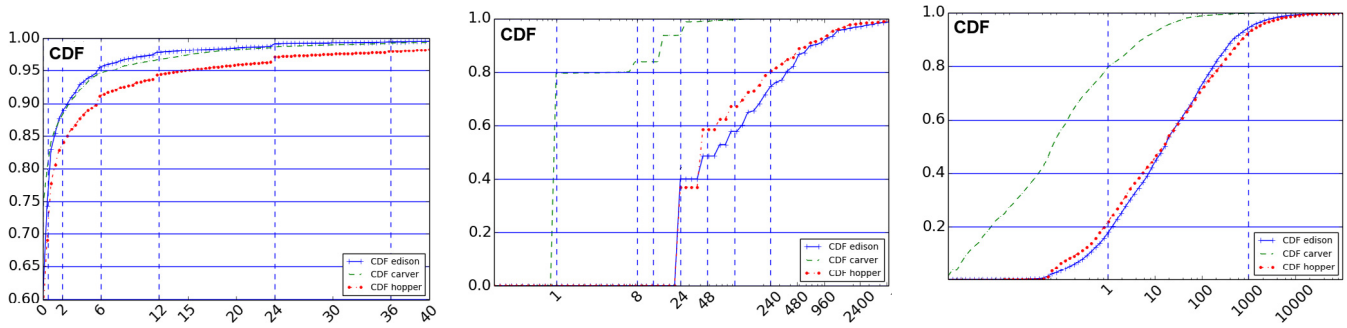
In this section, we describe the three systems analyzed (their characteristics, workload, scheduling model, and configuration), our data source (size, time span, format), analysis framework (motivation for analyzed variables), and our trend analysis methodology.

3.1. System descriptions

In this section, we describe the characteristics of the three systems to understand the context of our results.

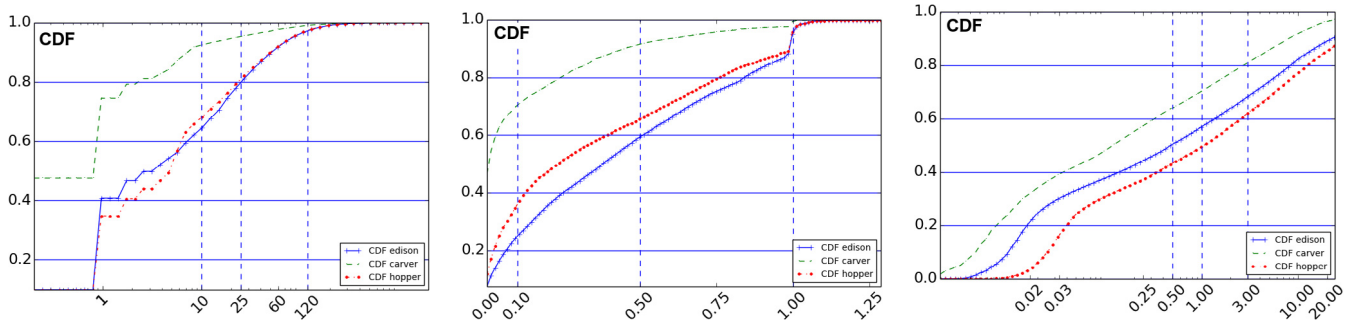
3.1.1. System characteristics

NERSC is an 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 Department of Energy (DOE). Our work analyzes job logs from three NERSC systems — Carver, Hopper, and Edison. These three systems were selected due to their diverse hardware characteristics and origin in the timeline of HPC systems evolution (Table 1). Carver is a IBM



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

Fig. 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 used 240 cores or fewer. (c) Carver's jobs use significantly fewer core-hours.



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

Fig. 2. Job characterization on Hopper, Edison, and Carver. (a) Carver receives significantly more job submissions per unit time 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 execute: Edison (67%), Hopper (60%), Carver (79%).

iDataPlex Linux cluster [30] 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 is a petascale Cray XE supercomputer, based on AMD processors and a Gemini interconnect [37], and deployed in 2010. Edison is a newer, more power efficient petascale Cray XC30, constructed with Intel processors supported by an 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.

For resource management, all three systems use the Moab scheduler [8,10,30] running atop the Torque resource manager [34]. 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 [3,4]. 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.

Carver provides a *serial* queue [28]. The serial queue allows users to submit and execute jobs that need a single core. Carver has 80 compute nodes allocated to serial jobs. Serial queues were added to Hopper and Edison in late 2014. The serial queues on Edison and Hopper are configured to use the Cluster Compatibility Mode (CCM). There are 15 compute nodes allocated to run serial jobs on Edison and Hopper. Serial queues contain jobs running long time (limited to 48 h) on a single core. The purpose of the serial queue is to increase resource utilization density. It packs jobs on the same node that do not benefit from parallelism and where performance is either not critical or rarely affected by resource sharing.

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

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. For example, jobs allocation sizes will cluster around allowed values in submission queues. We present the configuration of the system 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 [39]. 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 h of run time maybe submitted to the same submission queue, to be sorted into two execution queues with ranges of [0, 5), [5, 10) run time hours. Table 2 shows queue properties that govern the scheduling decisions for our three systems.

Table 2
Hopper, Edison, and Carver queue characteristics. Jobs have to be within certain limits to be accepted in a queue: requested run time 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.

Hopper Queues	Wall clock	Cores	E.	P.	Edison Queues	Wall clock	Cores	E.	P.	Carver Queues	Wall clock	Cores	E.	P.
bigmem	24 h	1-8,856	1	0						matgen_low	Unk	1–256	66	0
ccm_int	30 m	1–12,288	2	1	cm_int	30 m	1–12,288	2	1	matgen_prior	Unk	1–256	66	10
ccm_queue	96 h	1–12,288	16	1	ccm_queue	96 h	1–16,368	16	0	matgen_reg	Unk	1–256	66	1
debug	30 m	1–12,288	2	1	debug	30 m	1–12,288	2	1	debug	30 m	1–256	1	2
low	48 h	1–16,392	6	−3	low	48 h	1–16,392	6	−3	low	24 h	1–256	3	−2
premium	48 h	1–49,152	1	2	premium	36 h	1–49,152	1	2	xlmem_sm	72 h	8	1	0
reg_1 h	1 h	Unk.	8	0	reg_1 h	1 h	Unk.	16	0	xlmem_lg	72 h	32	2	0
reg_big	36 h	49,153–98,304	2	1	reg_big	36 h	49,153–98,304	2	1	reg_big	24 h	257–512	1	0
reg_long	96 h	1–1,536	4	0						reg_long	168 h	1–128	1	0
reg_med	36 h	16,369–49,152	4	1	reg_med	36 h	16,369–49,152	8	1	reg_med	36 h	129–256	2	0
reg_short	6 h	1–16,368	16	0	reg_short	6 h	1–16,368	24	0	reg_short	4 h	1–128	4	0
reg_small	48 h	1–16,368	16	0	reg_small	48 h	1–16,368	24	0	reg_small	48 h	1–128	3	0
reg_xbig	12 h	98,305–146,400	2	0	reg_xbig	12 h	98,305–131,088	2	1	reg_ xlong	504 h	1–32	1	0
										interactive	30 m	1–64	1	2
thruput	168 h	1–48	500	0	killable	48 h	1–16,368	8	0	serial	48 h	1	20	–

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 submits 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 queue characteristics on jobs wait time is presented in Section 6.2.

3.1.4. Queue configuration

The scheduler re-calculates each job's priority depending on the queue selected during its submission. We present the queue configurations in detail since they affects the job ordering and overall system behavior.

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 execution order of the jobs. These policies changed slightly through the period of our study; we present 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 and 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 queues have access to different sets: *matgen* queue jobs can only run on *matgen* resources (but jobs from other queues can use the matgen queue 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 resource core-hours available. We study the impact of this characteristic on job wait time and results are detailed in Section 6.2.

The *serial* queue jobs allocate one core per job and are executed on multi-job nodes (more than one job per node) that are exclusive to this queue. The wait time behavior of the *serial* queue is not compared to other queues since it does not compete on the usage of its resources.

3.2. Data source

All workload analysis is performed on the job summary entries from the 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 constitute the data pipeline to process the log data. This data pipeline is divided into three components. First, a data extractor, which retrieves the log files from the NERSC repository, parses them, eliminates invalid entries and inserts them in a MySQL database. Second, 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 analysis toolkit 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 scientific libraries SciPy and NumPy combined with the plotting library MatPlotLib [18]. 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 variables studied under the job perspective are:

Job size includes wall clock time (requested and effective), degree of parallelism, and resulting compute time allocation. These parameters define the system boundary requirements and job granularity.

Wall clock time accuracy represents the accuracy of the user estimations on the job run time. 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 and system sizes, a smaller inter-arrival time represents a larger job load and longer job wait times are expected.

Job diversity measures the difference between the geometries of jobs in the workload. 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 queue perspective includes the study of:

Queue significance represents the impact of each queue on the overall system. It provides insight on the impact of the properties of each queue to the overall system behavior according to their importance.

Queue job diversity captures the similarity between the jobs within each queue in terms of geometry. This analysis is relevant because the queue-priority system is the mechanism used by the scheduler to prioritize jobs depending on their geometry and importance. If this mechanism is not correctly aligned with the workload characteristics (e.g., the existing queues do not represent the most significant geometries present in the job mix) the scheduler might fail to bring the system to the state specified in its configuration. In our work, we are focused on two objectives (a) understanding the diversity of the jobs across the entire workload, and (b) 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 provides insight on the effects of the job diversity, job geometry, and queue configuration on the effectiveness of the queue-priority mechanism.

3.4. Trend analysis

The workloads of Carver and Hopper were analyzed for each year of their lifetime but this paper only details the results of 2014. The relevant results of all years were aggregated in the trend analysis to present the evolution of Carver and Hopper workloads through their lifetime. Edison's lifetime was too short at the time this analysis was performed to capture the trend analysis.

In the trend analysis, the evolution of the system's workload, overall performance, and user behavior are presented. 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 projected by analyzing the workload in sequential time periods and aggregating the results over a period of time, an adequate period had to be chosen. The size of the workload periods is calculated by detecting repeated user patterns in the workloads through Fourier transform analysis on the number of tasks submitted per hour [38].

Table 3
Detailed job characteristics distribution analysis.

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%

4. Job characterization

In this section, we provide the characterization of job geometry, user submission patterns, and job diversity on Edison, Hooper, and Carver in 2014.

4.1. Job geometry

The Cumulative Distribution Function (CDF) analysis of the job variables provides information on the patterns in resource allocations and dominant job groups that the scheduler manages. The absence of smaller jobs can predict low system utilization as backfilling scheduling requires such jobs to function efficiently. We present the analysis results for each of the job variables in this section.

Job wall clock time. Fig. 1a shows the Cumulative Distribution Function (CDF) of the job wall clock time for the three systems in 2014, re-framed to show jobs that run up to 40 h. Table 3 summarizes the most relevant values observed in the CDF.

The run time of the jobs is skewed towards short run times. In the case of Hopper, although we observe jobs running up to 160 h, a high percentage run for less than two hours (88%). This is not exclusive to Hopper, in fact, 86–88% of the jobs on all three systems run for fewer than two hours. In Carver, jobs are even shorter since numerous run times are shorter than one hour and 60% of them are shorter than 13 min.

Additionally, all three CDFs present *bumps* around 30 min and 6, 12, 24, and 36 h. These job durations are similar to some of the queue wall clock time limits (similar across three machines as seen in Table 2). These *bumps* might suggest that a significant group of jobs are submitted with a run time equal to maximum allowed in their corresponding queue.

Cores per job. Fig. 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). However, few cores are requested for most of Hopper's jobs i.e., 75% allocate 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, with significantly smaller allocations. On Carver, many jobs run on few cores i.e., 99% run on 240 cores or fewer, and 92% of all jobs run on a single node.

Allocated core-hours per job. Fig. 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 observe that nearly 10% of Edison and Hopper jobs consume more than 1000 core-hours individually.

4.2. Job characteristics

In this section, we study the relationship of job variables to user's submission patterns (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 wait time is presented in Section 6.2.

Inter-arrival time. Fig. 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. This analysis also helps us 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 s (25–33 min) range. On the other hand, more than 95% of Carver's inter-arrival times are under 25 s. 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 be preempted. Values significantly over one correspond to preempted jobs where run time is small compared to the extra time required to kill them. The exact time when a job gets preempted is also dependent on the time between scheduling passes.

Fig. 2b shows the distribution of the wall clock time accuracy values for the three systems. The initial steep slope of Carver's CDF shows that it executes many jobs that use much less than the requested wall clock time. Edison and Hopper have linear CDF for values between zero and close to one. However, in all systems there are numerous jobs with an accuracy slightly above one. The percentage of such jobs 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. Fig. 2c presents the distribution of job wait times under 24 h (jobs with longer wait times are not included in this graph). The figure shows that Hopper has jobs with longer wait times compared to Edison and Carver. Considering all the jobs in the system, 61% of Hopper jobs, 67% of Edison jobs, and 80% of Carver jobs support 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 for data clustering (*k*-means). We use and extend previous work on job grouping in cloud workloads [25]. The analysis aims to calculate the smallest possible number of *k*-means clusters [15] among the job geometry tuples with a maximum variation coefficient (v.c.), that is standard deviation divided by the mean, of 1.1. If jobs are largely similar, the method will group them in few clusters. Likewise, jobs from more diverse workloads will be grouped in numerous clusters.

As an initial step, jobs are transformed into geometry tuples composed by two real numbers — the job's actual wall clock time and the number of cores it allocates. Next, data tuples are normalized or "whitened" [7] to reduce the effect of the magnitudes of the values on the clustering process. The obtained tuple set is then analyzed with the minimum clusters search algorithm presented in Fig. 3.

```
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
```

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

Minimum clusters search algorithm. The *k*-means algorithm receives an input dataset and *k* centroids used as search starting conditions to produce *k* clusters. However, those *k* clusters are not guaranteed to be the minimum possible of not dispersed clusters. As a consequence, *k*-means must be invoked with different *k* sizes and different start conditions to explore the space of possible clusters in search of the minimum cluster set. Fig. 3 illustrates our search algorithm that uses a heuristic to calculate *k-means* initial conditions from previous clustering results.

First, the algorithm whitens the input job geometry tuples (line 5) to reduce the effect of the value sizes on the clustering. Next, the process to find the minimum number of *k*-means cluster (lines 7–37) is repeated 10 times with two random tuples (different each time) as initial centroids. In each attempt (lines 10–37), starting at $k = 2$, the algorithm calculates *k* clusters (line 14). Each cluster is analyzed (lines 15–23) and if it is not dispersed (v.c. $<$ 1.1), it is left as it is and its centroid is used in the next clustering. If a cluster is *dispersed* (v.c. $>$ 1.1), it is split, its centroid is removed, and two new centroids close the original one are added (line 21) to the initial condition of the next clustering. After the centroids analysis-split ends, the clustering is repeated with the new list of centroids as starting condition. This process is repeated until non dispersed clusters are found (lines 26, 27) or *k* is larger than a previously observed *k* of a successful search (lines 11,12). This algorithm does not guarantee that the absolute minimum number of clusters is found. However, it produces a good enough local minimum.

Diversity analysis results. Fig. 4 shows the results of the clustering search method for Edison's jobs in 2014. This graph is a scatter plot 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

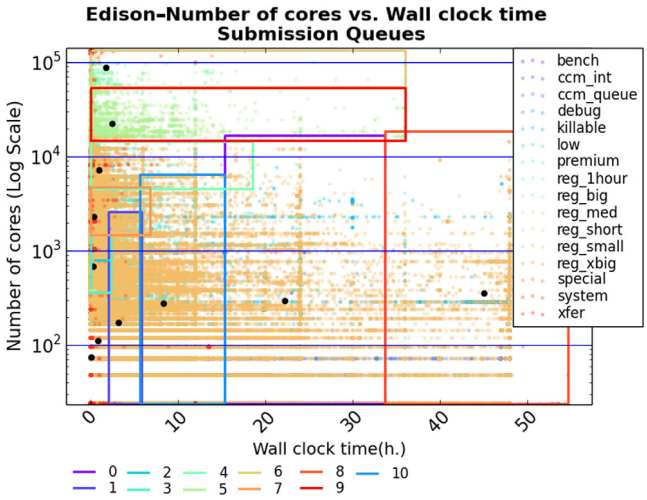


Fig. 4. Result of the job clustering method for Edison 2014 with 11 clusters. Jobs are mapped on queues and clusters: Each dot is a job and dot color indicates the queue. Black dots are cluster centroids and color boxes are the surrounding jobs belonging to the same cluster. Clusters are sets of jobs with similar geometry. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

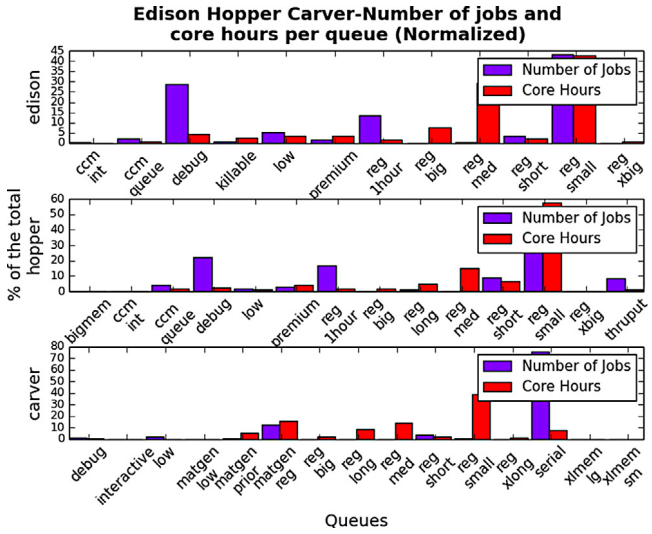


Fig. 5. Normalized view of the number of jobs and core-hours per queue in Edison, Hopper, and Carver.

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 centroids of the clusters 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 6 shows the results of the clustering. Eight clusters were found for Carver, 11 for Edison and 12 for Hopper. The lower number of clusters in Carver indicates a more homogeneous job set than the ones from Edison and Hopper (which are similar in their diversity). Carver's homogeneity could be explained by the significant presence of jobs from the *serial* queue in its workload (70% of total jobs). *Serial* jobs are not very diverse — they allocate a single core and their run times are all close to the queue run time limit.

5. Queue characterization

In the considered systems, job priorities and eligible job geometries are defined by the execution queue properties. The queues configuration settings influence the user submission behavior, the distribution of job priorities, and the resulting overall system behavior. In this section, relationship between queue configuration, job geometry, and user submission behavior is analyzed.

Analyses in this section focus on the execution-queues and the interpretation of the obtained results is based on the description of queues based on Table 2.

5.1. Queue significance

The significance of each queue in the system is analyzed to understand its impact on the overall behavior. The significance of a single queue is measured under three criteria: percentage of jobs and core-hours contributed to the system by the queue and its overall function. Fig. 5 shows the normalized view (in percent) of the number of jobs and core-hours contributed by each execution queue to the workloads of Edison, Hopper, and Carver. The role of each queue is presented in Section 3.1.3.

On Hopper, the queues *reg_small* (30%), *debug* (20%), *reg_1hour* (15%), *thruput* (9%), and *reg_short* (8%) contribute the largest number of jobs to the workload. In terms of core-hours, the order changes slightly, with *reg_small* (56%) still being the most significant queue but followed instead by *reg_med* (∼ 15%), and *reg_short* (∼ 7%). The shift in the ordering roots in the size and use of jobs that they contain. For example, *debug* contains many jobs that consume few core-hours because it is used to test small job executions before running larger jobs. The short wait times in the debug queue are managed by using a pool of debug-only resources, jobs having higher than average priority (1) while allowing large CPU allocations, but limiting the maximum run time to 30 min. The *reg_small* queue in Hopper is significant because of its contributed jobs and core-hours. It is followed by *debug*, which consists of small but many jobs that are necessary for testing before running successful larger jobs in other queues.

Edison queues are very similar to Hopper's. By contributed jobs, the order of significance is *reg_small*, *debug*, *reg_1h*, *low*, and *reg_short*. In terms of core-hours, the order is *reg_small*, *reg_med*, and *reg_big*. Similar to Hopper, the two most important queues are *reg_small* (many jobs and many core-hours), and *debug* (many jobs and important function).

Carver queues present a different significance pattern. By number of jobs, the *serial* queue is the most significant (70%) followed by *matgen_reg* (∼ 10%), and *reg_short*. From the point of view of core-hours, the ordering is *reg_small*, *matgen_reg*, *reg_med*, and *serial*. There are two special cases in Carver. First, the *serial* queue was created to run all the single core, long run time jobs at NERSC, that are important part of the Carver workload. The serial jobs run on exclusive resources (80 nodes) and they do not interfere with other jobs wait time. Second, Carver's *debug* queue is not significant in terms of number of jobs or core-hours. This is an interesting difference that could be explained by the longer user experience and confidence with Carver (an older system in 2014) and possibly simpler hardware configuration. For the rest of the queues, we can consider *reg_small* the most important queue since it contributes ∼ 40% of the workload core-hours.

5.2. Queue diversity

The job diversity analysis presented in Section 4 helps us understand the dominant job groups present in the workload of Edison, Hopper, and Carver. These groups could be used as queue templates to do precise prioritization of jobs by their geometry if workload configuration was known a priori. In this section, we

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].

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			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

analyze how the detected dominant job groups map on the existing queues and how they deviate from the ideal queue template.

Individual queue homogeneity. As a first step, we define the *queue dominant cluster* as the job cluster which contributes most jobs to a queue. We define a *queue homogeneity index* as the percentage of queue jobs belong to the queue's dominant cluster. For example, a queue sources jobs from three clusters with 20%, 30%, and 50% shares. The dominant cluster contributes 50% of the jobs and, as consequence, the *queue homogeneity index* is 0.50. A higher metric value indicates that many jobs of the queue are mapped to the same cluster and thus, the queue contains more self-similar jobs. A lower metric value indicates that the queue's jobs are more heterogeneous, and it is further from the ideal template established by the clusters. Also, findings presented later in Section 6.2 suggest that a low homogeneity index might be associated with non uniform wait time behaviors within a queue.

Table 4 presents the queue homogeneity indices for all the queues in Edison, Hopper, and Carver. In the case of Hopper, lowest queue homogeneity indices appear in *bigmem* (0.31), followed by *reg_small* (0.36), *reg_short* (0.39), and *premium* (0.40). These queues contain diverse jobs and are candidates to subdivide into smaller, better defined queues. In particular, an analysis needs to be performed on the *reg_small* queue. It is Hopper's most significant queue (in contributed core-hours and jobs), and scores a very low homogeneity index.

Edison queues are slightly more uniform than Hopper's. Although the *premium* queue scores a smaller homogeneity index (0.27) than any of Hopper's, all the rest score over 0.40. Like in Hopper, the *reg_small* should be studied as its index is low (0.42) and it is the most significant queue in the system.

Carver queues are numerous and a few have low homogeneity indices. *xlmem_lg* (0.24) is the least homogeneous followed by *reg_small* (0.26), *debug* (0.32), and *interactive* (0.35). Again, the importance of the *reg_small* and its low homogeneity index point to a need to consider dividing the queue. In contrast, Carver's *debug* queue is much less homogeneous (0.32) than the two other systems (0.63, 0.70). It could be attributed to the small presence of *serial* jobs, very different from regular jobs but without the overwhelming presence and homogeneity of their native queue. It is necessary to perform further studies on the range of debug applications run in all systems to fully understand this difference and significance.

Table 5
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.

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

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 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 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 work does not contain system availability information. Thus, we calculate the *theoretical Total System Utilization* (tTSU) as:

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

where *time period* is the number of hours in the studied time period and *maximum system capacity* is the total number of CPU cores of the system. By definition, tTSU will be less than or equal to TSU. We present the reported TSU and the tTSU in Table 5. Carver's reported TSU was not available for 2014.

6.2. Job wait time

Previous analysis in Section 4.2 presents a coarse grained analysis of job wait time. Such analysis is important but limited, since it does not analyze the impact of factors such as job geometry and priority on wait time. Since representing and analyzing the impact of three job variables (i.e., requested time, requested CPU cores, and priority) on wait time is complex, we divide this analysis in two steps. First, we start with the two variables that are observed to be more significant and calculate the job's median wait time grouped by queues (and thus corresponding priority) and requested number of CPU cores. Second, we calculate the median job's wait time grouped by requested CPU cores and run time in each of the queues.

Wait time vs. requested CPU cores and priority. We present job wait time for the three systems as heat maps in Fig. 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 h 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.

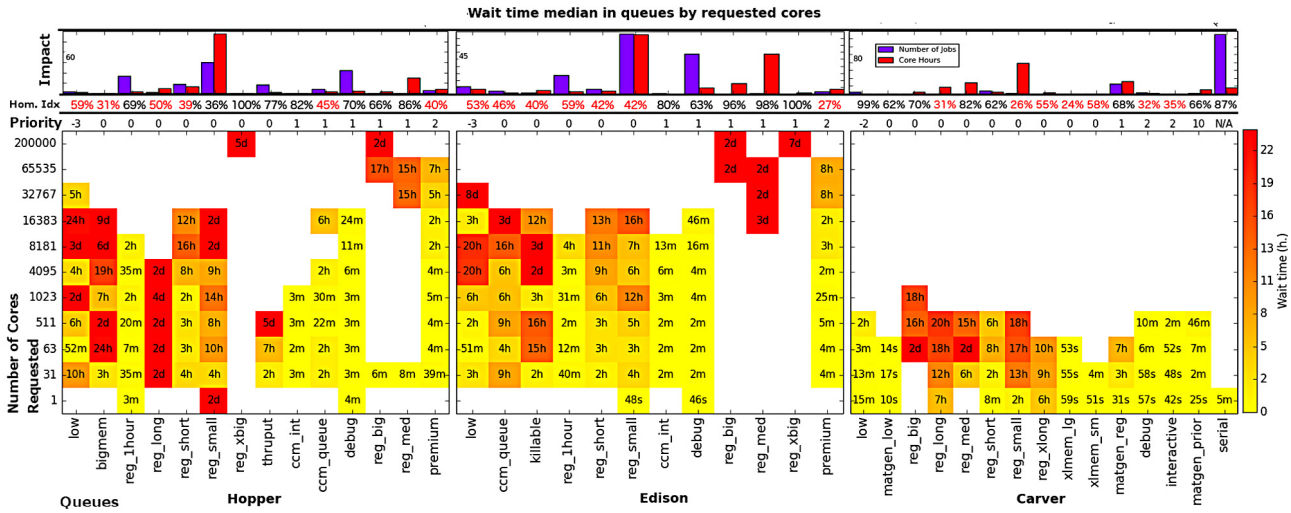


Fig. 6. Job wait time median per queue depending on the requested cores. Aggregated on top: priority per queue, median of the wall clock time of queue jobs, jobs per queue (normalized), core-hours per queue (normalized).

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*, show the expected behavior. These priority zero queues have longer wait times than the ones with priority one and shorter than the one with −3. Queues *bigmem*, *reg_long*, *reg_small*, and *thruput*, present wait times significantly higher. The *bigmem* queue is the gateway for the nodes with more memory (384 large memory nodes vs. 6000 regular nodes) and its jobs 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 smaller number of cores allocated). In some cases, it could be related to the job 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 for both the queues.

Edison exhibits a similar behavior to Hopper. The queues *killable* and *ccm_queue* have longer wait times, because the jobs are 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 time for its jobs. The jobs with higher priorities behave as expected, showing shorter wait times for similar job sizes.

Carver displays different trends compared to Hopper and Edison. Its *serial* queue has exclusive resources and a median wait time of five minutes. The *matgen_low*, matgen_reg, and matgen_prio queues have a pool of resources, but those might be used by other queues. Carver's *xlmem* queue is similar to Hopper's *bigmem*, and meant to serve jobs with large memory requirements. However, the resource mapping is different on Hopper. On Hopper the *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 on the special nodes. This exclusive access, combined with *xlmem*'s low job count and core-hour contribution results in the median job wait time being 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.

Wait time vs. requested CPU cores and run time in each queue. In this section, we present an analysis of job wait time depending on requested resources, requested run time, and priority. Although analyses were performed for all queues in each system, only the results for six queues of Hopper are presented: *low*, *bigmem*, *reg_1hour*, *reg_small*, *debug*, and *premium*. They contain low (*low*), medium (*bigmem*, *reg_1hour*, *reg_small*), and high (*premium*) priority and have different homogeneity indices (from 27%-*premium* to 70%-*debug*). Our results showed that the queues of Edison and Carver are equivalent to the ones presented here.

Fig. 7 shows six heat maps that represent the median wait time of jobs grouped by requested CPU cores and run time in six queues of Hopper. For each queue, the *x*-axis represents intervals of requested run times, adapted to the observed values (e.g., in *big_mem* interval size is 60 min since observed run times span from less than 60 min to 50 h, while in *reg_1hour* the interval size is 5 min since observed values span from less than 5 min to one hour). The *y*-axis represents a non-linear categorization of the possible number of cores allocated to jobs. The median wait time values are represented in tones from bright yellow (close to 0) to dark red (24h or more) like in Fig. 6. Brighter colors (shorter wait times) are expected in the lower left corner (smaller allocation, shorter run time) and darker (longer wait times) on the top right of each queue wait time map (larger allocation, longer run time). Maps of queues with a higher priority should contain brighter tiles (thus shorter wait time) for similar requested run times.

The results in Fig. 7 indicate that queues with lower homogeneity indices show more discontinuities in their job wait times, with *premium* (homogeneity index 27%) presenting the noisiest heat map. In decreasing noisiness, *premium* is followed by *big_men*

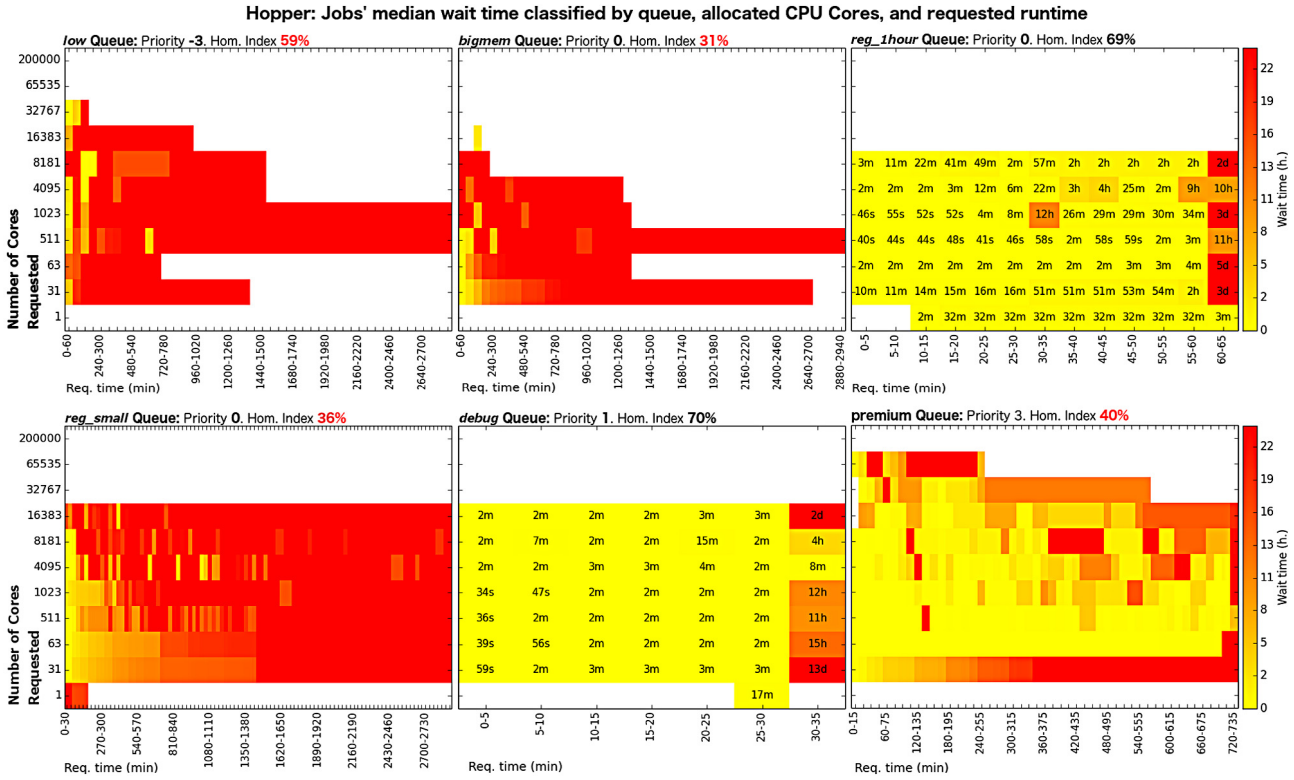


Fig. 7. Job wait time median per queue depending on the requested cores and run time in six queues of Hopper including their corresponding priority and homogeneity indices. Similar analyses were performed also for Edison and Carver. Queues with lower homogeneity index present more disturbances in the color gradient and thus in the expected wait time depending on a job's requested CPU Cores, run time, and priority.

(31%), *reg_small* (36%), and *low* (59%). The two queues with higher homogeneity indices present more continuous heat maps. Queue *reg_1hour* (69%) presents minor discontinuities in intermediate job sizes and the upper bound of the job run time. Queue *debug* (70%) only presents discontinuities in the longest running jobs in the queue (around 30 min).

Concerning the comparison across queues and priorities, a more dominant yellow color (shorter wait times) for similar job geometries is present in heat-maps of higher priority queues. Correspondingly, a darker color (longer wait times) is present for similar jobs in lower priority queue maps. However, queues with lower homogeneity indices present multiple discontinuities in this pattern. For example, the *premium* queue, with priority of three, presents many of these discontinuities: jobs requesting 64 to 511 CPU cores for 135 to 150 min show significant longer run times than in the *reg_small* queue (priority 0). Similar discontinuities appear in all the maps, being more present in the less homogeneous queues like *premium*, *big_mem*, or *reg_small*.

In summary, the results in Figs. 6 and 7 show that low homogeneity disrupts the expected wait time behavior of the jobs in the three systems. Also, wait time does not increase as expected as jobs' allocate more CPU cores, for longer times, or have lower priority. In the future, it will need to be considered to see if more predictable wait times are possible by dividing these queues according to the observed job clusters, increasing the per-queue homogeneity index.

6.3. Queue homogeneity

Analyses in Section 6.2 show that queues with low queue homogeneity might present disturbances that make job wait time hard to predict. In this section, we explore the combined effect of this phenomena as it might happen across many queues of a system. For example, if all the queues of a system have very low homogeneity indices, independently of its geometry or queue, a job's wait time will be hard to predict. In the other extreme, if all the queues are very homogeneous, a job's wait time should be very easy to predict according to its geometry and priority. The following metrics intend to assess the position of a system between this two example extremes.

We create two overall metrics by combining the queue homogeneity indices with the two quantitative queue significance criteria presented in Section 5.1 (queue's contributed jobs and core-hours) to measure the overall queue homogeneity. The first metric is **Job homogeneity index**, calculated as a linear combination of all the queue homogeneity indices. The coefficients are the share of jobs contributed by the corresponding queue. For example, Queue1 has a homogeneity index of 0.6 and contributes 30% of the system's jobs. Queue2 has an homogeneity index of 0.4 and contributes 70% of the jobs. The *Job homogeneity index* is thus calculated as: $0.6 \cdot 0.3 + 0.4 \cdot 0.7 = 0.46$. A lower index indicates that many jobs are in queues with low homogeneity indices and thus many job's wait time might be affected. If many jobs are affected, wait time increase will likely be more noticeable by majority of the users.

The second metric is the **Time homogeneity index** , also calculated as a linear combination of the queues homogeneity indices, but under the core-hours significance perspective. The coefficients are the shares of core-hours contributed to the system by the corresponding queues. 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: $0.6 \cdot 0.8 + 0.4 \cdot 0.2 = 0.56$. The time homogeneity index is an overall measure of the effect of wait time disturbance on the core-hours processed in the system. A lower index indicates that many core-hours correspond to jobs in queues with low homogeneity indices. This effect could be more notable

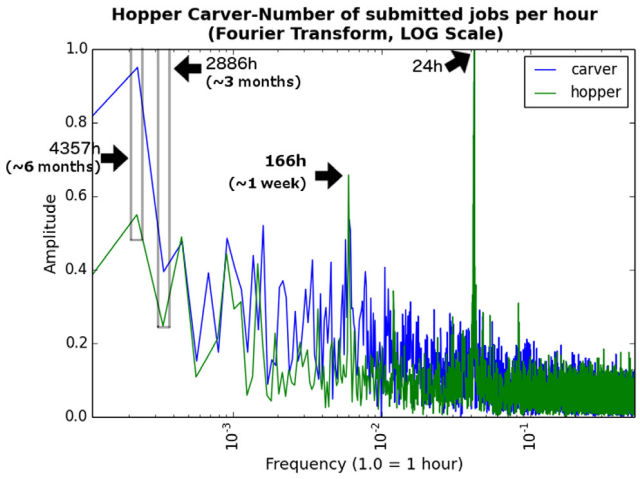


Fig. 8. Fourier decomposition of the series of the jobs submitted per hour for Hopper and Carver to detect dominant submission cycle. Note the logarithmic scale for the frequencies. Most powerful frequencies highlighted with a black arrow and its corresponding period.

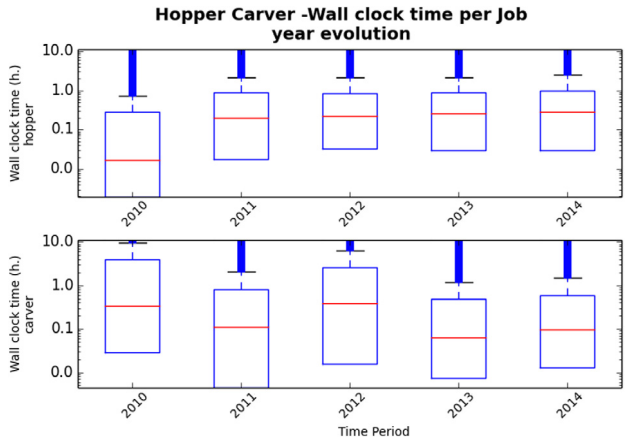


Fig. 9. Box-plots of the job wall clock times observed in each workload year. Trend: Hopper jobs become longer, Carver jobs shorter. Majority of jobs under one hour.

Table 6
Job and queue homogeneity analysis results: for each machine, minimum number of *k*-mean clusters discovered in the jobs and overall homogeneity indices.

	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

to power-users (large job submitters) and system administrators, specially if the job homogeneity index is high.

Table 6 shows the calculated *homogeneity indices* for the three NERSC systems in 2014. Carver presents a high job index, which indicates that many of its jobs are in uniform queues. This matches the fact that most of its jobs are in the serial queue, with very simple uniform wait times. Also, its time index is 0.5, which indicates that the jobs which belong to heterogeneous queues are large. This is confirmed by the 0.26 homogeneity index in its most relevant queue in terms of core-hours (*reg_small*).

Edison shows the highest time homogeneity index. This suggests that Edison is the system with more core-hours from jobs in homogeneous queues. This is confirmed by the fact that two of the largest queues in terms of core-hours(reg_big, and reg_med) are very homogeneous. Hopper and Edison have the similar job homogeneity indices in the range of 0.5. However, Hopper time index is lower, which suggests that its larger jobs are contributed by heterogeneous jobs, and thus might be subject to wait time increases.

7. Trend analysis

In this section, we present a lifetime analysis of the various workloads parameters of Hopper and Carver from Jan 2010 to June 2014. The purpose is to observe if there are any clear evolution patterns over the life cycle of these HPC systems.

7.1. Time patterns and analysis granularity

First, we choose a time period to slice the data by performing a pattern analysis of the data to understand the administrative realities of the system. User behavior patterns were detected by studying the Fourier spectrum [38] of the number of batch jobs submitted per hour in two years. The result is presented in Fig. 8, where black arrows point to the most powerful frequencies that correspond to the periods of 1 day, 1 week, 3 months, and 6 months. The daily and weekly repeating patterns correspond to user working periods, in which users tend to submit more jobs during work hours (daily pattern) and work days (weekly pattern) [6]. The three and six months pattern are related to the allocation rules set by NERSC. Each project has a number of core-hours to be used in a year, divided in 4 allocation quarters (three months) 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 time period for the trend analysis.

7.2. Job geometry

The evolution of the actual job run time variable is presented in Fig. 9 as a box plot of the values registered for each system in each year. Hopper shows a significantly low wall clock time median in 2010 ($<$ 1 min), a year in which this system was a smaller testbed. In 2011, the median increased to $\sim$ 5 min and subsequently increased to $\sim$ 12 min by 2014. Carver shows a different trend: the median and upper and lower quartiles decrease effectively over the period studied. The median decreased from $\sim$ 20 min (2010) to $\sim$ 6 min (2014). However, during their lifetime the relationship of the two systems changed. In the first year in production, Carver ran longer jobs than Hopper. However, this changed in 2014, when Hopper ran longer jobs that Carver. Overall, Hopper and Carver present fairly short jobs and the highest upper quartiles in both systems is around the one-hour value.

The evolution of the job width (number of allocated cores per job) is presented in Fig. 10. For Hopper, the median decreases from 100 cores (2010) to under 30 cores (2014). Carver presents a very different pattern. Except for 2010, its median is one core, indicating the predominance of single core serial jobs. In 2014, the upper quartile increased to 8 cores. In 2010, Carver's jobs are more parallel than in the rest of the years because its *serial* queue did not exist (added in 2011). When the queue was added, Carver was identified as the system to run serial jobs (allocating a single CPU core) at NERSC, increasing the presence of such jobs in the workload.

The core-hours allocated by jobs were also analyzed and the results are presented in Fig. 11. In the case of Hopper, per job allocated core-hours remain almost unchanged through time with a median of $\sim$ 20 core-hours in all years and the upper quartile slightly under 200 core-hours in most years. In the case of Carver, it slowly decreases from a median of nearly 1 core-hour to $\sim$ 6 core minutes (and a last upper quartile of 1 core-hour).

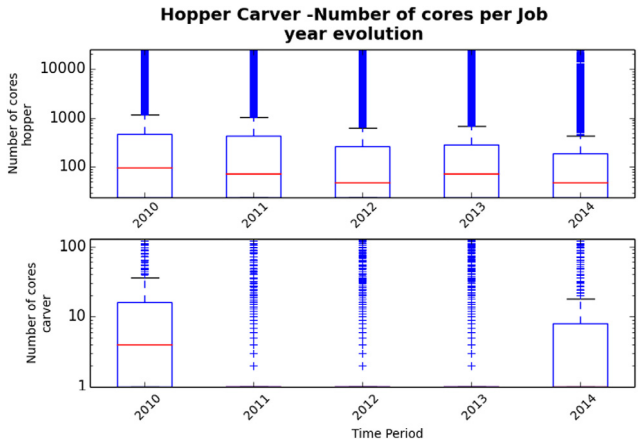


Fig. 10. Allocated number of cores for each workload year. Trend: Hopper jobs allocate fewer cores. In 2011–2013, most Carver jobs used one core.

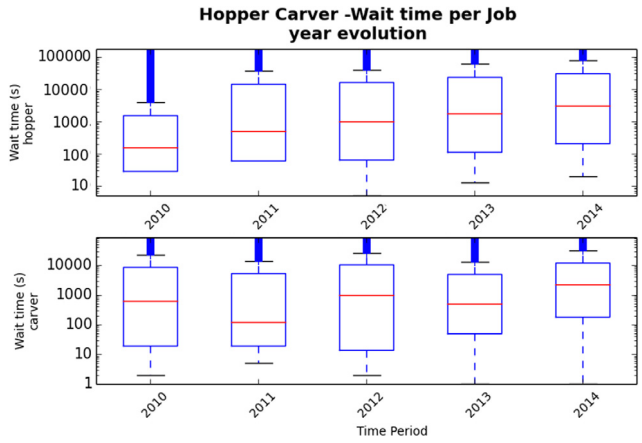


Fig. 12. Jobs' wait time evolution for each workload year. Trend: All systems increase wait time. Carver lower wait time in 2011.

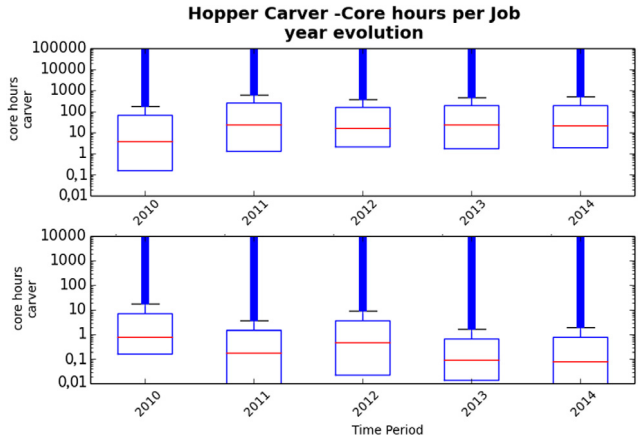


Fig. 11. Allocated core-hours for each workload year. Trend: No changes on Hopper. Carver jobs become smaller.

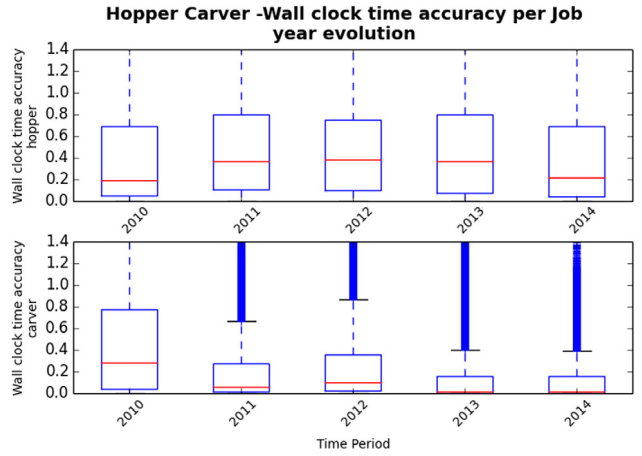


Fig. 13. Jobs' wall clock time accuracy evolution for each workload year. In all systems wall clock time remains low.

In summary, Hopper jobs (shorter jobs, with a higher degree of parallelism, bigger than Carver's job) seem to be showing an increase in their wall clock time. Since the core-hours per job remain similar, this indicates that jobs request less CPU cores. Carver jobs (longer jobs, lower degree of parallelism, fewer core-hours than Hopper's) have decreasing wall clock time and request more CPU 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 Fig. 12, the median wait time of Hopper jobs steadily increases from under 100 s to over 20 min (a constant growth also present in the upper and lower quartiles). In Carver, the wait time increases in a zig-zag pattern from $\sim$ 10 min (2010) to $\sim$ 20 min (2014). In 2011, Carver has significantly shorter wait times, which is likely due to an increase in compute resources of the system. The steady increase of wait time over the lifetime could be attributed to the growth of the number of users.

7.4. Wall clock time accuracy

As presented in Fig. 13, Hopper does not show a clear trend. On Hopper, 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 ($\sim$ 0.25) and 2011 ($<$0.1). In 2014, the median is under 0.1 and the last quartile is under 0.2. For Carver, estimation quality clearly decreases over time. In general, both systems present very low values with medians under 0.4. These values indicate that through the life of both systems, the decisions made by the backfilling algorithms are based on inaccurate user estimations.

7.5. Job and queue diversity

Following the methodologies presented in Sections 3.3, 4.3, and 6.3, the diversity analysis was performed for each year and system. We studied the evolution of the overall jobs diversity (number of dominant job groups) and overall queue diversity (time/job homogeneity indices). Results are presented in Fig. 14.

Hopper shows fewer dominant job groups (clusters) over time, decreasing from 17 to 12 clusters, implying a reduction in the job geometry diversity. In the case of Carver, the job mix is fairly homogeneous (7 clusters) except in the second year (13 clusters), a year in which hardware updates were performed. Carver presents a more homogeneous job mix in comparison to Hopper over the lifetime of the systems.

As presented in Fig. 14, Hopper's *job homogeneity index* increases from 0.36 to 0.71. This might indicate an evolution towards significantly more jobs contributed by homogeneous queues, which could improve the overall wait time. The *time homogeneity*

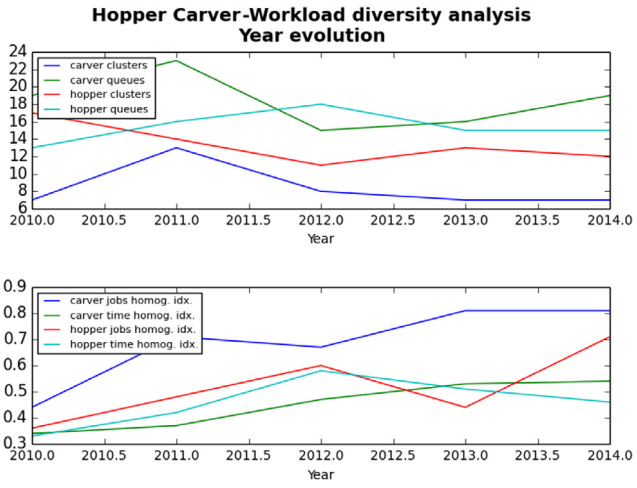


Fig. 14. Workload diversity and in queue homogeneity index: Overall workload becoming less diverse. Job mixes in queues are becoming more uniform.

index increased from 0.33 to 0.46, which points to a small increase of the homogeneity in queues relevant in terms of contributed core-hours. However, in 2014, the high job index value combined with low time index value might indicate significant unpredictability in the large job wait times. Caver shows a similar pattern with a large increase in terms of evolution of the *job homogeneity index* (from 0.44 to 0.81) and a smaller increase for the time index (from 0.34 to 0.54). Also, there is a larger increase in *job homogeneity index* in 2011, when the *serial* queue was added.

As we compare both systems, Carver indices are generally larger indicating that most relevant queues of this system tend to be more homogeneous than that of Hopper.

8. Summary

In this section, we summarize our results from our analyses.

8.1. Summary of a year's workload

We summarize the key results from the detailed analysis performed on the 2014's workload from Edison, Hopper, and Carver. We also compare them with preexisting analyses of similar HPC systems.

- The job wall clock times are short on all three systems: 86% to 88% of the jobs run less than 2 h.
- On Edison and Hopper, 37%, 39% of jobs run on one node and 69%, 75% run on 10 nodes or fewer. On Carver, 92% of the jobs run on a single node.
- On Carver, 77% of its jobs allocate one or less core-hours. Carver jobs use far fewer hours than jobs on Hopper and Edison.
- 60% of Edison jobs, 66% of Hopper jobs and 95% of Carver jobs run less than 50% of their requested time. On the other hand, jobs run over their estimated wall clock time in the case of 10% Hopper's jobs, 11% of Edison's and 8% of Carver's.
- Carver has the most homogeneous workload (more similar jobs, and more homogeneous dominant queues). Hopper has a diverse workload with a complex job mix in its queues. Edison sits between the two.
- Carver jobs suffer the longest wait times, although they allocate significantly fewer core-hours than jobs on Hopper and Edison.
- On all systems, disturbances in the job's wait times have been observed in queues with low homogeneity indices. This reduced the predictability of the wait time on the systems. According to the overall time and job homogeneity indices, Carver should be the more predictable, followed by Edison and Hopper.

8.2. Comparison with other systems

Although these results represent the state of current systems, it is important to understand their difference to other similar systems. In particular, we compare our results to analyses on Intrepid and Stampede (characteristics summarized in Table 7), one past and one current HPC system.

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. Intrepid is more similar to Carver in its configuration. 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, which is different from the NERSC systems that has more powerful cores. We compare with Intrepid trace of nine months from 2009 [11].

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 similar to Edison or Hopper. In fact, Edison's processor are the next generation (Ivy Bridge) to Stampede's (Sandy Bridge). We compare our results with a previous analysis over a trace of three months of Stampede in 2013 [9].

Edison and Hopper's wall clock time distribution matches the patterns observed on Intrepid [11]. Carver's run time CDF is steeper and similar to Stampede [9]. Jobs on 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 similar 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 [1] which is not considered in the scope of this paper.

8.3. Workload evolution

Observing the evolution of large scientific infrastructures through their lifetime provides insight on the evolution of the systems as their use matures. It also provides data to understand and extrapolate the characteristics of future workloads.

Fig. 12 shows that the wait time steadily increases as the systems age. This trend fits with a growing scientific community using the same system and more advanced applications that require faster infrastructure. There is one exception, in 2011 Carver presented significantly smaller wait times possibly due to its expansion.

As Hopper and Carver are compared in Figs. 9 and 10, the analysis on the evolution of the job geometry reveals that Hopper jobs (which had shorter jobs but with a higher degree of parallelism than Carver) seems to be increasing their wall clock time but using

Table 7
Intrepid and Stampede characteristics.

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

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.

Fig. 14 shows that 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 natural 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 [27]. Second, the serial batch queue (long-running, low-degree of parallelization) was added [29]. Carver's workload became uniform towards the end of its life.

The diversity analysis study was partly motivated by the need to understand the effect of application diversity on the workload jobs, but the results show that jobs became more uniform as the systems age. The explanation is that new application effects on diversity are neutralized by the overwhelming presence of classical HPC application which motivates the existence of the system itself. For example, as described in site NERSC reports [4], more than 75% of Hopper's core-hours are consumed by a few algorithms classical in the HPC world (e.g., fusion simulation, Lattice QCS, density function theory, climate and Geo simulation, or molecular dynamics). It is also likely that non-traditional applications face challenges on the systems that affects their usage patterns. Thus, it is difficult to predict the exact distribution of the workload patterns on future HPC systems. Our results will be an important reference point as HPC systems evolve to accommodate the needs of diverse workloads including stream processing and high throughput jobs.

9. Conclusions

In this paper, we present methods to analyze NERSC's workload. We develop a methodology that includes traditional workload analysis techniques (e.g., CDF analysis of job variables) and incorporates new methods to assess job heterogeneity. The job heterogeneity analysis includes a novel algorithm that employs *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 us to assess the effect of job heterogeneity on the scheduling performance in terms of wait time.

Our evaluation establishes a reference of the state of the workload in 2014 of three high performance systems (Edison, Hopper, and Carver). These results can help understand the behavior in current similar HPC systems, including: (1) The job geometries are fairly diverse including significant number of smaller jobs compared to older systems. (2) The low per queue homogeneity indexes, show that single priority policies are affecting jobs with a fairly diverse geometry. (3) The wait time analysis shows that studied queues with low homogeneity indices present poor correlation between job's wait time and geometry. (4) Job submission patterns show that the accuracy of user predictions of their job's wall clock time (fundamental for the performance of backfilling functions) is very low, and does not improve over time. (5) Hopper and Carver workloads present a clear trend in their four year lifetime, i.e., 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 provide a strong foundation for future scheduling research and systems operations management. Scheduling research needs to address present and future workloads. Our work provides important insights in understanding characteristics of future systems (e.g., diverse jobs, smaller jobs, or low accuracy in run time 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 in finding policies to encourage users to provide better predictions. Better run time 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 result in hard to predict wait times. Queues obtained by subdividing dominant job groups are expected to show predictable wait times.

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, both 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 Union's 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] 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.

[2] 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.

[3] K. Antypas, NERSC-6 Workload Analysis and Benchmark Selection Process, Lawrence Berkeley National Laboratory, 2008.

[4] K. Antypas, B.A. Austin, T.L. Butler, R.A. Gerber, NERSC Workload Analysis on Hopper, Tech. Rep., LBNL Report: 6804E, 2014.

[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] N.-C. Chen, S. Poon, L. Ramakrishnan, C.R. Aragon, Considering time in designing large-scale systems for scientific computing, in: Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing, ACM, 2016, pp. 1535–1547.

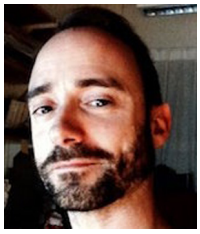
[7] A. Coates, A.Y. Ng, Learning feature representations with k-means, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 561–580.

[8] T.M. Declerck, I. Sakrejda, External Torque/Moab on an XC30 and Fairshare, Tech. Rep., NERSC, Lawrence Berkeley National Lab, 2013.

[9] J. Emeras, Workload Traces Analysis and Replay in Large Scale Distributed Systems, (Ph.D. thesis), Grenoble INP, 2014.

[10] Y. Etsion, D. Tsafrir, A Short Survey of Commercial Cluster Batch Schedulers, Vol. 44221, School of Computer Science and Engineering, the Hebrew University of Jerusalem, 2005.

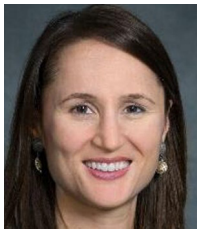
[11] D. Feitelson, Parallel workloads archive 71 (86) (2007) 337–360. http://www.cs.huji.ac.il/labs/parallel/workload.
[12] D.G. Feitelson, Workload Modeling for Computer Systems Performance Evaluation, Cambridge University Press, 2015.
[13] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, Parallel job scheduling, a status report, in: Job Scheduling Strategies for Parallel Processing, Springer, 2005, pp. 1–16.
[14] E. Frachtenberg, D.G. Feitelson, Pitfalls in parallel job scheduling evaluation, in: Workshop on Job Scheduling Strategies for Parallel Processing, Springer, 2005, pp. 257–282.
[15] J.A. Hartigan, M.A. Wong, Algorithm AS 136: A k-means clustering algorithm, Appl. Stat. (1979) 100–108.
[16] T. Hey, S. Tansley, K.M. Tolle, et al., The Fourth Paradigm: Data-Intensive Scientific Discovery, Vol. 1, Microsoft Research Redmond, WA, 2009.
[17] N.E. Huang, Z. Shen, S.R. Long, M.C. Wu, H.H. Shih, Q. Zheng, N.-C. Yen, C.C. Tung, H.H. Liu, The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 454 (1998) 903–995.
[18] J.D. Hunter, Matplotlib: A 2D graphics environment, Comput. Sci. Eng. 9 (3) (2007) 90–95.
[19] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. Epema, The grid workloads archive, Future Gener. Comput. Syst. 24 (7) (2008) 672–686.
[20] D. Jacobsen, NERSC Site Report, One year of Slurm, in: Slurm User Group, 2016. https://slurm.schedmd.com/SLUG16/NERSC.pdf.
[21] M.D. Jones, J.P. White, M. Innus, R.L. DeLeon, N. Simakov, J.T. Palmer, S.M. Gallo, T.R. Furlani, M. Showerman, R. Brunner, et al., Workload analysis of blue waters, 2017. ArXiv Preprint ArXiv:1703.00924.
[22] C.B. Lee, Y. Schwartzman, J. Hardy, A. Snavely, Are user runtime estimates inherently inaccurate? in: Job Scheduling Strategies for Parallel Processing, Springer, 2005, pp. 253–263.
[23] D.A. Lifka, The ANL/IBM SP scheduling system, in: Job Scheduling Strategies for Parallel Processing, Springer, 1995, pp. 295–303.
[24] U. Lublin, D.G. Feitelson, The workload on parallel supercomputers: modeling the characteristics of rigid jobs, J. Parallel Distrib. Comput. 63 (11) (2003) 1105–1122.
[25] A.K. Mishra, J.L. Hellerstein, W. Cirne, C.R. Das, Towards characterizing cloud backend workloads: insights from google compute clusters, ACM SIGMETRICS Perform. Eval. Rev. 37 (4) (2010) 34–41.
[26] NERSC, 2015-01-18. http://www.nersc.gov.
[27] NERSC, Magellan batch queues on Carver. 2015.01.15. http://www.nersc.gov/REST/announcements/message_text.php?id=1991.
[28] NERSC, Queues and polices (Carver). 2014.1.15. https://www.nersc.gov/users/computational-systems/carver/running-jobs/queues-and-policies/.
[29] NERSC, Serial queue on Carver/Magellan. 2015.01.15. http://www.nersc.gov/REST/announcements/message_text.php?id=2007.
[30] NERSC, Submitting batch jobs (Carver). 2015.1.15. https://www.nersc.gov/users/computational-systems/carver/running-jobs/batch-jobs/.
[31] 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.
[32] S. Schlagkamp, R. Ferreira da Silva, W. Allcock, E. Deelman, U. Schwiegelshohn, Consecutive job submission behavior at mira supercomputer, in: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, ACM, 2016, pp. 93–96.
[33] S.N. Srirama, P. Jakovits, E. Vainikko, Adapting scientific computing problems to clouds using mapreduce, Future Gener. Comput. Syst. 28 (1) (2012) 184–192.
[34] G. Staples, TORQUE resource manager, in: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, ACM, 2006, p. 8.
[35] D. Tsafrir, Y. Etsion, D.G. Feitelson, Modeling user runtime estimates, in: Workshop on Job Scheduling Strategies for Parallel Processing, Springer, 2005, pp. 1–35.
[36] D. Tsafrir, Y. Etsion, D.G. Feitelson, Backfilling using system-generated predictions rather than user runtime estimates, IEEE Trans. Parallel Distrib. Syst. 18 (6) (2007) 789–803.
[37] 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.
[38] W.W.-S. Wei, Time Series Analysis, Addison-Wesley Publ, 1994.
[39] J. Weinberg, A. Snavely, Symbiotic space-sharing on sdsc's datastar system, in: Job Scheduling Strategies for Parallel Processing, Springer, 2007, pp. 192–209.
[40] N. Zakay, D.G. Feitelson, Preserving user behavior characteristics in trace-based simulation of parallel job scheduling, in: IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, MASCOTS, IEEE, 2014, pp. 51–60.



Gonzalo P. Rodrigo Álvarez postdoctoral fellow at the Data Science and Technology Department in the Lawrence Berkeley National Lab. He completed his Ph.D. in Computing Science in 2017 at the Distributed Systems group in Umeå University (Sweden). At the same time, he was working as an external affiliate to his current Department at the Berkeley Lab. He spends his life understanding how machine learning can help indexing scientific datasets, solving HPC workflows scheduling problems, understanding data intensive cloud workflows scheduling challenges, and implementing simple scheduling techniques (yet powerful) on Slurm. Rodrigo also spent a summer at Google Inc. Originally from Spain, previously, he worked for almost ten years in the online gaming, and optical network industries.

P.-O. Östberg is a Research Scientist with a Ph.D. in Computing Science from Umeå University and more than half a decade of both academic research and postgraduate industry experience. He has held Visiting Researcher positions at Uppsala University, Karolinska Institutet, the University of Ulm, and the Lawrence Berkeley National Laboratory (LBNL) at the University of California, Berkeley; and has worked in the Swedish government's strategic eScience research initiative eSSENCE as well as in several projects funded by the EU and the Swedish national research council (VR).

Erik Elmroth s a Full Professor and the Leader of the Distributed Systems research group at Umeå University. His background covers a broad spectrum of HPC, grid-, and cloud infrastructure research topics. Prof. Elmroth has been Chair of the Swedish National Infrastructure for Computing (SNIC), a member of the Swedish Research Councils Committee for Research Infrastructures, as well as Chairman of its expert group on science infrastructures, and has written two research strategies for the Nordic Council of Ministers. Recognition for his research includes the Nordea Scientific Award and the SIAM Linear Algebra Prize.



Katie Antypas is the Department Head of Scientific Computing and Data Services at the National Energy Research Scientific Computing (NERSC) Center. She has served various roles over the years including NERSC Services Department Head, Project Lead for the NERSC-8 system procurement, group leader for User Services at NERSC, and co-implementation team lead on the Hopper project. Before coming to NERSC, Katie worked at the ASC Flash Center at the University of Chicago supporting the FLASH code, a parallel, adaptive mesh refinement astrophysics application. She has an M.S. in Computer Science from the University of Chicago and a B.A in Physics from Wellesley College.

Richard Gerber is Senior Science Advisor and High Performance Department Head at the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory in Berkeley, CA. Richard has a Ph.D. and M.S. in Physics (Computational Astrophysics) from the University of Illinois at Urbana–Champaign and held a National Research Council Postdoctoral Fellowship at NASA Ames Research Center before jointing NERSC in 1996. He has more than 30 years experience in high performance scientific computing.

Lavanya Ramakrishnan is a Staff Scientist at Lawrence Berkeley National Lab. Her research interests are in software tools for computational and data-intensive science. Ramakrishnan has previously worked as a research staff member at Renaissance Computing Institute and MCNC in North Carolina. She has Master's and Doctoral degrees in Computer Science from Indiana University and a Bachelor's degree in Computer Engineering from VJTI, University of Mumbai. She joined LBL as an Alvarez Postdoctoral Fellow in 2009.