LLNL-CONF-634532

A Comparative Study of High-Performance Computing on the Cloud

A. Marathe, R. Harris, D. Lowenthal, B. de Supinski, B. Rountree, M. Schulz, X. Yuan

April 11, 2013

ACM Symposium on High-Performance Parallel and Distributed Computing
New York, NY, United States
June 17, 2013 through June 21, 2013

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

A Comparative Study of High-Performance Computing on the Cloud

Aniruddha Marathe
Rachel Harris
David K. Lowenthal
Dept. of Computer Science
The University of Arizona

Bronis R. de Supinski
Barry Rountree
Martin Schulz
Lawrence Livermore
National Laboratory

Xin Yuan
Dept. of Computer Science
Florida State University

ABSTRACT

The popularity of Amazon's EC2 cloud platform has increased in recent years. However, many high-performance computing (HPC) users consider dedicated high-performance clusters, typically found in large compute centers such as those in national laboratories, to be far superior to EC2 because of significant communication overhead of the latter. Our view is that this is quite narrow and the proper metrics for comparing high-performance clusters to EC2 is *turnaround time* and *cost*.

In this paper, we compare the top-of-the-line EC2 cluster to HPC clusters at Lawrence Livermore National Laboratory (LLNL) based on turnaround time and total cost of execution. When measuring turnaround time, we include expected queue wait time on HPC clusters. Our results show that although as expected, standard HPC clusters are superior in raw performance, EC2 clusters may produce better turnaround times. To estimate cost, we developed a pricing model—relative to EC2's node-hour prices—to set node-hour prices for (currently free) LLNL clusters. We observe that the cost-effectiveness of running an application on a cluster depends on raw performance *and* application scalability.

1. INTRODUCTION

In recent years, Amazon's Elastic Compute Cloud (EC2) platform has had significant success in the commercial arena, but the story for high-performance computing (HPC) has been mixed. While "success" stories appear in the popular press periodically, most of them feature an embarrassingly parallel program being run on tens of thousands of cloud machines [8]. A more complicated issue is how well EC2 performs on more tightly-coupled applications, which are more representative of applications HPC users typically execute on HPC clusters (e.g., those at national laboratories or other supercomputing centers). The prevailing opinion is that EC2 is essentially useless for such applications [18, 33].

There are reasons to justify skepticism of EC2 for tightly-coupled, more traditional HPC applications. First, the latency and bandwidth of the network used by EC2 are usually inferior to that of a typical, dedicated HPC cluster (e.g., Ethernet vs. Infiniband, although Infiniband cloud offerings seem likely in the near future [32]). Second, compute nodes are virtualized, which causes concerns in terms of virtualization overhead as well as virtual machine co-location.

However, to compare EC2, which provides a fee-for-service model in which access is essentially available 24/7, to traditional HPC clusters on only the axis of execution time is unfair. This comparison ignores, for example, the sometimes significant queue wait time that occurs on HPC clusters, which typically use batch scheduling. Of course, it also ignores factors such as cost, where HPC clusters have a significant advantage. After all, HPC clusters in supercomputing centers such as Livermore Computing (LC) at Lawrence Livermore National Laboratory (LLNL) are free to the user, even though this is only an artifact of government funding.

In this paper, we take a novel look at these differences and we contrast high-end Amazon EC2 clusters against traditional HPC clusters, *but with a more general evaluation scheme.* First, we compare EC2 to five LLNL HPC clusters based on total turnaround time for a typical set of HPC benchmarks at different scales; for queue wait time on the HPC clusters, we use a distribution developed from simulations with actual traces. Second, to enable a comparison on total cost of execution, we develop an economic model to price LLNL clusters assuming that they are offered as cloud resources at node-hour prices. Because at reasonable scales, cloud computing platforms guarantee zero queuing delay, we disregard waiting time while modeling prices. Using well-known methods in economics on commodity resource pricing, the model achieves profit maximization from the perspective of the cloud provider. Using these node-hour prices, we then compare EC2 with the LLNL HPC clusters and argue that from a cost effectiveness perspective, applications should be mapped to the most appropriate cluster, which is not necessarily the highest performing one.

We make the following contributions in this paper.

- We evaluate EC2 and HPC clusters along the traditional axis of execution time at reasonable scales (over 1000 cores).

- We develop a pricing model to evaluate HPC clusters in node-hour prices based on system performance, resource availability, and user bias.

HPDC'13, June 17–21, 2013, New York, NY, USA.
Copyright 2013 ACM 978-1-4503-1910-2/13/06 ...$15.00.

- We evaluate EC2 and HPC clusters along more general axes, including total turnaround time and total cost. This provides to the best of our knowledge the first comparison of EC2 and HPC clusters from a *user perspective* (as opposed to a data center perspective [33, 35]).

Our results show that the decision of whether to choose EC2 or HPC clusters is complicated. First, EC2 nodes are high-end and, thus, performance is comparable to HPC clusters on EC2 for some applications that incur modest communication. However, we confirm prior results showing that communication intensive applications are typically inefficient on EC2 [33]. Second and more importantly, while HPC clusters usually provide the best execution time, queue wait time on these frequently oversubscribed resources can lead to much larger turnaround times. For example, when median wait time is exceeded, total turnaround time on the LLNL machines is often larger than that on EC2 (sometimes by more than a factor of 4) even though execution time on LLNL machines (once the application starts) can be several times faster. Finally, using the modeled node-hour prices, we show that the choice of most cost-effective cluster for an application is non-trivial and depends on application scalability as well as user bias of cost versus turnaround time. For example, for a cost bound of $5000 per hour, the optimal cluster choice for LAMMPS is EC2, leading to a 51% lower turnaround time than the fastest cluster. On the other hand, given a 90 second time bound, it is 8.2% cheaper to run LU on Hera (an LLNL machine) than the fastest cluster.

The rest of this paper is organized as follows. Section 2 provides background of EC2 and motivates our comparison of EC2 to HPC clusters. Section 3 describes the machines and provides our experimental setup. Section 4 provides comparison of raw system performance. Section 5 provides an evaluation based on queue wait times and turnaround times, and Section 6 compares based on our pricing model. Section 7 discusses the implication of our results. We provide related work and our conclusions in Sections 8 and 9.

2. BACKGROUND AND MOTIVATION

The term *cloud computing* is somewhat difficult to define precisely. A traditional definition is that it provides, at an actual monetary cost to the end-user, computation, software, data access, and storage that requires no end-user knowledge about physical location and system configuration [31]. From a high-performance computing (HPC) perspective, the cloud provides a choice of different clusters. Each cluster potentially provides different resources: number and type of cores, amount of memory, storage, and network latency and bandwidth. In this paper, we assume homogeneous computing, though we realize that certain cloud providers may not always make this guarantee for all of their clusters.

2.1 EC2 Basics

We focus on the most popular cloud platform, which is Amazon EC2 [6]. Amazon sells several kinds of *virtual machine* instances (VMs), which comprise cluster nodes. A virtual machine is an isolated, guest operating system that exists within a host system. There can be many virtual machines in one physical machine, and consequently a virtual machine has *resources*, as defined by an instance, that can be up to, but not exceeding, the resources on the physical machine.

Amazon EC2 markets several different instances, which are distinguished by different computational and network capabilities. In this paper we focus on the highest-end instance, called "cluster compute eight extra large", because it is the instance intended for HPC applications. EC2 also markets several kinds of ways of purchasing time on their systems; in this paper we use *on-demand* pricing, in which the user pays money for each VM instance and receives access to the purchased node immediately. We leave consideration of *reserved* and *spot-market* pricing for future work.

By default EC2 does not provide any guarantees of physical node proximity [14]. However, EC2 does allow physical proximity through a *placement group* on Cluster Compute. It is unclear how many nodes in a placement group a user can acquire without wait times similar to batch systems. In our experiments, we observed no delay due to placement groups, but we used at most 128 nodes. Compared with HPC systems, though, batch systems cannot guarantee physically proximate nodes either (they perform best effort [21]).

We emphasize that the cloud notion of having *no wait time* has limits. Executions on tens of thousands of cloud nodes [8], for one-time capability or hero type program executions, likely require some wait time or pre-arrangement with the cloud provider. These kind of runs are clearly better suited for resources as large compute centers, where control and scheduling is local and machines can be used for dedicated application times (note that such jobs will also not be handled by batch queuing systems any more and do require manual intervention or scheduling). In this paper, however, we focus on job sizes that are below that threshold and occur frequently, as those make up a vast majority of HPC workloads, in particular production workloads. Those jobs can be easily handled by the pooled resources of cloud providers and therefore require no wait time.

2.2 Comparing EC2 to HPC Clusters

As mentioned above, this paper compares EC2 and HPC clusters using turnaround time and cost; the latter necessitates developing a pricing model for HPC clusters. Even if HPC clusters are "free", the user may consider a cloud cluster for the following reasons[1]. First, the application may be compute intensive, and some of the nodes offered by EC2 may execute such applications faster than HPC nodes, as cloud providers typically can afford a faster upgrade/refresh cycle for their machine park. Second, the application may execute faster from actual start time to finish on an HPC cluster, but the total turnaround time on the cloud may be less because of wait queue delay on the HPC clusters.

The second point above must be tempered by cost. That is, if we expand our notion of execution time to a less traditional metric such as total turnaround time, we cannot ignore the cost difference between an EC2 node (significant) and an HPC node ("free"). On the other hand, the HPC node is not really "free", and for a fair cost comparison, we need to create a pricing model.

Thus, there is a trade-off if one evaluates an EC2 cluster versus an HPC cluster on the basis of cost and performance. It depends on many factors, including the application, the current cluster utilization, and the cost per node. The goal

[1] In this paper, we do not consider the trivializing case where a user does not have access to an HPC cluster.

of this paper is to try to characterize these factors and to better understand in which situations using EC2 for traditional HPC applications makes sense. Note that in this paper, we do not make any judgments about the relative importance of turnaround time and cost.

3. EXPERIMENTAL SETUP

This section describes our experimental setup and test platforms. First, we provide a description of all test systems and benchmarks used in our evaluation. Second, we describe configurations used by the benchmarks.

3.1 Machine and Benchmark Description

Table 1 shows configurations for our test systems. Five of our systems reside at Lawrence Livermore National Laboratory (LLNL), which we refer to as "LLNL clusters" or "HPC clusters" interchangeably in the rest of the paper. Sierra and Cab are newer clusters at LLNL. Sierra consists of 1849 Intel Xeon 5660 with 12 cores per node, a clock speed of 2.8 GHz, 12 MB cache, and a memory size of 24 GB/node. Cab has 1296 Intel Xeon 2670 nodes with 16 cores per node, a clock speed of 2.6 GHz, 20 MB cache, and a memory size of 32 GB/node. Both Sierra and Cab have Infiniband QDR inter-node connectivity. Hyperion runs 1152 nodes with 8 cores per node, a clock speed of 2.5 GHz, 6MB cache, 12 GB/node system memory and Infiniband DDR inter-node connectivity. Currently, the largest partition available on Hyperion contains 304 nodes. Hera is a somewhat older system; it has 800 Opteron nodes with 16 cores per node, clock speed of 2.3 GHz, 512 KB cache, a memory size of 32 GB/node, and Infiniband DDR inter-node connectivity. LLNL also hosts uDawn, a BlueGene/P systems. It has 2048 nodes running IBM PowerPC processors with a clock speed of 850 MHz, 4 cores per node, 2 KB caches and a memory size of 2 GB/node.It is connected internally by a 3D torus network for point-to-point communication.

Amazon EC2 offers two HPC-oriented virtual machines: Cluster Compute Quadruple Extra Large "CC1" and Cluster Compute Eight Extra Large "CC2". In this paper we focus on the more powerful instance, CC2, which consists of Xeon Sandy Bridge processors with two oct-cores, a clock speed of 2.59 GHz, 20 MB cache, a memory size of 60.5 GB/node, and 10 Gb Ethernet inter-node connectivity. CC2 has hardware-assisted virtualization support to reduce overhead. For convenience we most often refer to this as simply "EC2" in the rest of the paper.

We use benchmarks from the NAS Parallel [7], ASC Sequoia [4], and ASC Purple [3] benchmark suites. Specifically, we run CG, EP, BT, LU and SP from the NAS suite; Sweep3D and LAMMPS from ASC Sequoia, and SMG2000 from ASC Purple. We did not execute all of the programs from a given suite because we wanted some diversity, and executing all of the benchmarks from each suite would have taken several extra hours of compute. The cost per hour at scale (128 nodes/1024 tasks) on EC2 is over $300.

3.2 Program Setup

We used MVAPICH-1.7 (for the LLNL clusters) and MPICH2 (for the EC2 clusters). We compile all benchmarks using the `-O2` option. All experiments avoid execution on core 0. This is because EC2 currently pins all interrupts on to core 0. For communication intensive programs, this causes severe load imbalance on core 0 and significant performance degradation, a problem first reported by Petrini [25]; and, we borrow their solution of leaving core 0 unused. We executed separate experiments that show that using core 0 leads to as much as a 500% overhead on our benchmarks. Personal communication with Amazon indicates that in the near future interrupt handling will be spread throughout the cores [27], and we expect that this problem will then cease to exist.

However, we go further and, in fact, use only *half* of the available cores on a node (except on Sierra, where we use 8 out of the 12 available cores because of the power-of-two nature of the benchmarks). We use only half of the cores because to use 15 out of the 16 cores on a node would lead to an uneven core distribution (e.g., 64 cores spread over 6 nodes with 15 utilized cores each and one node with 4 cores). This can cause additional communication and imbalances in the applications due to the particular topology mapping chosen by individual MPI implementations (which are different between the LLNL clusters and the EC2 clusters). Our experiments show that on a core-to-node mapping that is a power-of-two, this additional communication is minimized. We also disabled hyperthreading on the EC2 cluster, because our experiments showed that hyperthreading most often degrades performance.

We use strong scaling, so in each set of results, all of the benchmark sizes are identical across different MPI task counts, and we configure the benchmarks to run for between 30 and 170 seconds on EC2 across all scales. For the NAS programs, we edited `npbparams.h` directly; the benchmark sizes were close to class C (sometimes smaller, sometimes larger). For SMG2000, we use a size of 65x65x65 at 1024 tasks, and then adjusted sizes accordingly at lower scales to convert it to a strongly scaled application. For Sweep3d, we use the `makeinput` utility and modified the sizes. For LAMMPS, we use the Lennard-Jones input deck.

Our benchmarks cover a wide variety of message characteristics. Many are communication intensive (BT, CG, LU, and SP), sending, per MPI rank, at least 100K messages totaling at least 1 GB (SP sends over 2 GB per rank). SMG2000 sends about 400 MB per rank. LAMMPS sends about 200 MB per rank, but did so over far fewer messages (only about 1000 per rank), and has far less time spent in MPI communication than BT, CG, LU, SMG2000, or SP. Finally, EP is computation intensive, sending only about 1 KB per rank.

4. COMPARING RAW PERFORMANCE

This section describes our base performance evaluation of clusters at LLNL and EC2. First, we measure point-to-point latency and bandwidth between physical nodes. Second, we analyze computation performance of a single node on each cluster with single-task configurations of standard HPC benchmarks. Third, we evaluate execution times at scales of up to 1024 tasks with standard HPC benchmarks.

4.1 Raw Computation and Communication Performance Evaluation

We use a set of simple microbenchmarks to measure performance of individual system parameters. Our tests cover network latency and bandwidth between nodes as well as relative single node computation performance.

First, we present the results of network latency and bandwidth, using a standard ping-pong benchmark, in Table 2. The experiments show that Sierra has the least inter-node

Cluster	CPU speed (GHz)	Cache size (MB)	Memory size (GB)	Cores/Node	Interconnect Technology	Cost ($/Hour)
Sierra	2.8	12	24	12	Infiniband QDR	—
Hera	2.3	0.5	32	16	Infiniband DDR	—
Cab	2.6	20	32	16	Infiniband QDR	—
Hyperion	2.4	6.0	12	8	Infiniband DDR	—
uDawn	0.85	0.02	2	4	3D Torus	—
EC2	2.59	20	23	16	10 GigE	2.4

Table 1: System specification for our test systems

	Cab	Hera	Sierra	Hyperion	uDawn	EC2
Latency	1.61 μs	2.23 μs	1.58 μs	1.91 μs	2.92 μs	55.15 μs
Bandwidth	22.6 Gb/s	9.3 Gb/s	23.1 Gb/s	16.9 Gb/s	3.1 Gb/s	3.7 Gb/s

Table 2: Network latency and bandwidth for Cab, Hera, Sierra, Hyperion, uDawn and EC2

latency and the fastest bandwidth. This is due to the Infiniband QDR technology used for communication. uDawn employs a 3-D Torus network for MPI point-to-point communication with about 5 times higher latency and 3 times lower bandwidth compared to Infiniband QDR in Sierra. EC2 shows low variance in network bandwidth, confirming previous work [?]; however, the latency is at least 50 times higher than Sierra. These times are consistent with results reported elsewhere [35, 33].

To understand the computation power of the systems, we executed the benchmarks with a single MPI task. Table 3 shows the result of executing all of our benchmarks on one MPI task. Each machine is expressed in terms of average speedup over uDawn, which is the cluster with worst single-node performance. The highest performing node is Cab, and the next fastest is EC2; both are the same architecture with different processor speeds. We used the Stream benchmark [23] to measure the memory bandwidth and latency on both Cab and EC2 nodes. We found that Cab has 40% higher memory bandwidth and about 30% lower memory latency than EC2. This affected performance of some memory-bound benchmarks such as BT and CG by as much as 30%. Also, our EC2 cluster does not co-locate virtual machines [27] and, because as stated above we avoid core 0, there is no significant noise overhead.

We did not specifically perform tests to try to characterize the virtualization overhead, because we do not have identical, non-virtualized nodes with which to compare. However, as shown earlier, sequential performance on EC2 nodes is quite good, even if virtualization overhead exists (plus, there is hardware support to reduce it). Others have studied the impact of virtualization on HPC applications [34, 17].

4.2 Execution Time at Scale

In this section, we first present results of our MPI benchmarks that allow us to compare EC2, Cab, Sierra, Hyperion, Hera and uDawn. We use 128 nodes and a total of 1024 MPI tasks. Second, we provide scaling results from 256 tasks to 1024 tasks.

We first consider the difference in execution time for the various systems, i.e., the elapsed time from program start to program end. Figure 1 shows the median values collected during at least three runs on the systems (normalized to EC2 times, with a breakdown of relative computation and communication time shown also). For the most part, our results here are similar to execution time measurements collected by others [35, 18], in the sense that communication-intensive applications have significant overhead on the cloud due to the use of 10 Gb/s Ethernet instead of Infiniband[2]. LU performs 1.3-2.6 times worse on EC2 than LLNL clusters, except uDawn, due to communication time dominating program execution. Due to their similarity to LU in relative performance on EC2 and HPC clusters, we do not show numbers for BT, CG and SP (space considerations prohibit it later in the paper, so we omit them everywhere for uniformity). SMG2000 performs 1.9-4.6 times worse on EC2 than Cab, Sierra and Hyperion, and about 28% slower than Hera.

For EP, EC2 is slower than Cab by 12% and faster than other clusters. For Sweep3D, EC2 is the fastest cluster. Somewhat surprisingly, EC2 outperforms both Hyperion and Hera on LAMMPS, but is 68% and 29% slower than Cab and Sierra, respectively. While these three codes do have non-trivial communication at 1024 tasks, the superiority of the EC2 nodes compensates somewhat for its inferior communication infrastructure.

Out of all LLNL clusters, uDawn shows slowest performance for most of the benchmarks. This is because uDawn uses slower, more power efficient processors and employs slower interconnects than Infiniband; its strength is that the BlueGene design scales to large node counts. In case of compute-bound applications, uDawn performs 4.2-4.7 times slower than EC2 with 1K tasks, even though uDawn has negligible system noise.

5. TURNAROUND TIME COMPARISON

This section compares job turnaround times on HPC and EC2 clusters on a variety of job sizes and core/node counts. Turnaround time is the sum of execution time and job queuing delay. We evaluate execution time using standard HPC benchmarks at large scales. We evaluate job queuing delay by both real and simulated experiments at different task scales and sizes that represent real-world jobs.

5.1 Queue Wait Times

This section is concerned with comparing total turnaround times, which is the time between submission of the job and

[2] As mentioned earlier, Microsoft Azure has announced its intention to offer an Infiniband-based cluster [32], which will presumably have competitive performance with HPC clusters.

uDawn	Cab	Sierra	Hyperion	Hera	EC2
1.00	9.55	7.30	7.22	3.62	8.22

Table 3: Relative sequential performance (normalized to uDawn) for Cab, Sierra, Hyperion, Hera and EC2.

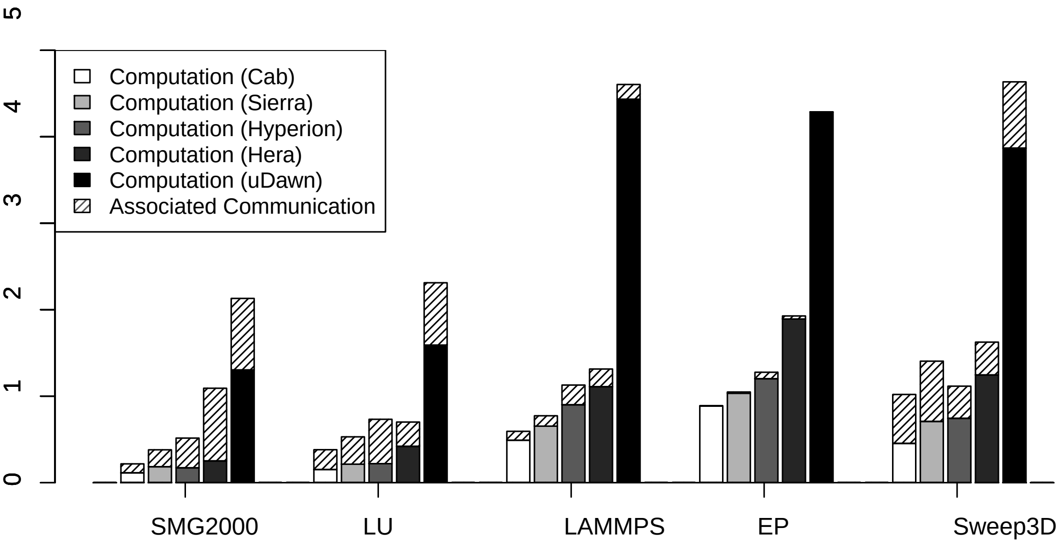


Figure 1: Comparison of execution times on Cab, Sierra, Hyperion, Hera, uDawn and EC2 clusters (128 nodes/1024 tasks). Times are normalized to those of EC2, and the relative percentage spent in computation and communication is shown.

completion of the program. LLNL clusters use batch submission and are optimized for execution time. On the other hand, EC2 optimizes for turnaround time [27].

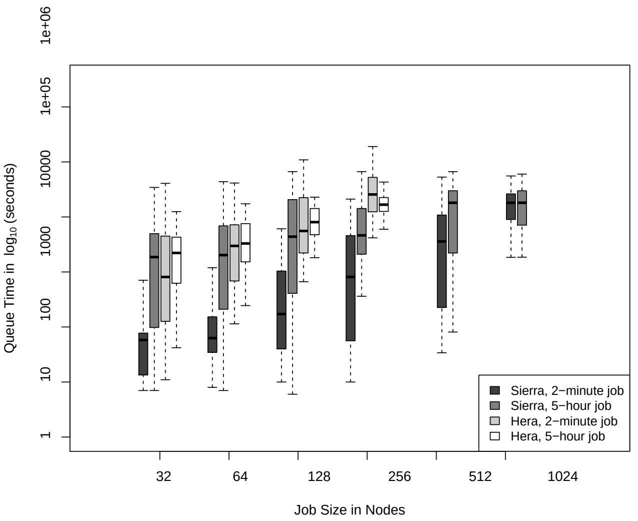
To measure turnaround time, we need to know queue wait time, which varies with HPC cluster, job size, and maximum run time. On EC2, using on-demand instances with non-excessively-sized requests, we observed low queue wait times. However, HPC clusters are well utilized by a large number of users and hence can have significant queue wait time.

Estimating wait time is particularly tricky because (1) the batch submission algorithm used is opaque, and (2) queue wait time may not be linear in the number of nodes or time requested. To estimate the wait time, we therefore simulate job execution times using the Alea-3 simulator [20]. Job scheduling on LLNL clusters is performed by the Simple Linux Utility for Resource Management (SLURM) job scheduler, which employs First-Come First-Serve and Backfilling algorithms [2]. We configure Alea-3 to match the SLURM properties using the Easy Backfilling algorithm, which is optimized for throughput, provided the maximum job execution time is specified at submission. Additionally, we modify the simulator to handle node-level allocations and output queue wait times. We use job logs for ANL's Intrepid cluster from the Parallel Workloads Archive [1] collected during January 2009 to September 2009. We chose Intrepid logs since it provided similar volume of workload compared to LLNL's Sierra, Hera, Cab, Hyperion and uDawn clusters. To estimate the wait times on each cluster, we configure the simulator separately with the machine specification for each LLNL cluster. For each cluster, we manually add jobs with a unique identifier at 12 hour intervals to probe for queue wait times in the simulation. These probe jobs were configured to measure wait times at task counts from 32 up to 1024 in steps of powers of 2. We set the maximum job time (the time at which, the program, if still executing, is killed) to (separately) 2 minutes and 5 hours (somewhat arbitrarily).

To validate our numbers, we also collect job wait times on Sierra and Hera at similar job sizes. The maximum job time was set to 2 minutes and 5 hours, and we submitted the jobs at 10 A.M. and 10 P.M. every day for two months. We would have liked to execute these experiments with a larger variety of maximum job execution times. However, we did not, as we wanted to minimize our consumption of LLNL resources.

The results are shown in Figure 2 as a series of boxplots, which show the median in addition to the ranges of each quartile. For real experiments, queue wait times increase with an increase in maximum job execution time, as long as a node request is below a significant percentage of the available nodes (which is roughly 64 nodes on Hera and between 512 and 1024 nodes on Sierra). In addition, the data shows that even for the two-minute jobs, there are potentially significant wait times.

Although the real clusters can have different workloads than the one used in our simulation, simulated runs show similar trends for queue wait times at increasing scales and job limits. For example, median values for simulated and real queue wait times for Sierra at 512 and 1024 tasks fall in the same range. Similarly, median simulated queue wait times for Hera at 128 and 256 tasks follow corresponding numbers in real experiments. Also, Sierra shows lower queue wait times than Hera in both simulated and real experiments, typically at higher scales, due to higher resource availability. Thus, the real queue wait times validate sim-



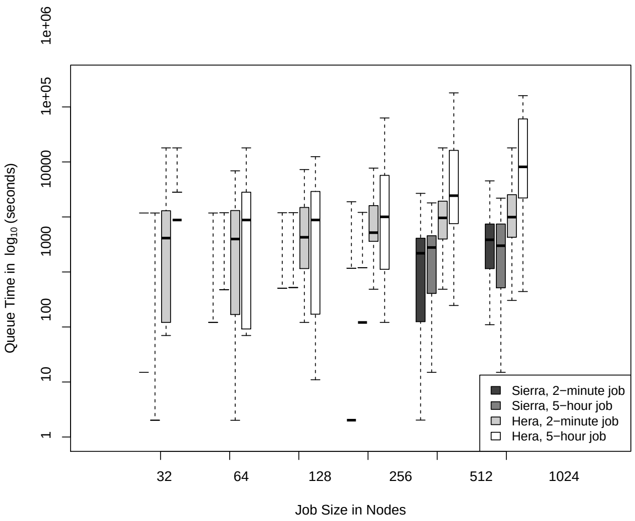


Figure 2: Boxplots showing comparison of (a) real queue wait times and (b) simulated queue wait times at different node counts on Sierra and Hera

ulated queue wait times at relevant scales. Hence, we use simulated numbers for other clusters for the analysis of total turnaround times.

On EC2, we measured queue wait time as the time from request submission to MPI hostfile setup on all nodes, because that is when an MPI program can be initiated. We measured wait times of 146 seconds, 189 seconds and 244 seconds to acquire 16, 32 and 64 EC2 nodes, respectively. Due to limited funds (each experiment to measure startup time on 64 nodes costs $150), we iterated our experiments few times, and our measurements are not statistically significant. However, because the wait times on EC2 are clearly orders of magnitude lower than those on LLNL clusters, this does not affect the validity of our overall findings.

5.2 Turnaround Time

In this section, we focus on the same six machines (EC2, Cab, Sierra, Hyperion, Hera and uDawn), but turn our attention to total turnaround time. Figure 3 presents a statistical representation of total turnaround time on all machines at three different MPI task counts: 256, 512, and 1024. The data is presented as follows. We scale the execution times on each LLNL cluster for an application on a 5-hour scale relative to execution times on EC2. The lower edge of the boxplot represents normalized execution time. Note that the fourth quartile of the boxplot is not visible because of the queue wait time distribution and normalization. We then add queue wait times on the respective clusters to get total turnaround times. Thus the combined plot shows the factor of total turnaround time for each application compared to EC2 total turnaround time. Because queue wait time is a distribution, we use a boxplot to represent it; here, we use the results we collected for the 5-hour wait times.

The results show two general trends: First, in many cases, the EC2 execution time is better at lower scales. For example, EP and LAMMPS on most clusters have higher execution times than EC2, especially at 256 tasks. In such situations, of course, total turnaround time will be much better on EC2, as queue wait time is an additive penalty on higher-end LLNL clusters. However, as we increase the number of MPI tasks, higher-end LLNL clusters scale better than EC2; again, this is not surprising as (1) we are using strong scaling, and (2) higher-end LLNL clusters use Infiniband, and EC2 uses 10 Gb Ethernet.

Second, at higher task counts, demand for more resources generally causes longer turnaround times. For example, consider applications LU and SMG2000 at 1024 MPI tasks. Clearly, the queue wait time governs turnaround time for most of the clusters (except uDawn, which consistently shows higher execution times than EC2). That is, for these applications, if the queue wait times fall within the first quartile, LLNL clusters are superior. On the other hand, if the queue wait times fall in the fourth quartile, then EC2 is clearly better. If the queue wait time falls in the second or third quartile, which system is better depends on (1) where in the quartile the wait time falls, along with (2) the relative superiority of the execution time on HPC clusters.

The effect of wait queue times is more pronounced for applications with overall higher computation times at higher scales, due to better execution times on EC2. For example, consider Sweep3D, EP and LAMMPS on LLNL clusters and EC2 at 1024 MPI tasks. In most cases, EC2 turnaround time (horizontal line) falls below HPC cluster execution times.

We can also view the expected wait time using existing methods such as QBETS[24]. With QBETS, a binomial is used to determine a confidence level that a given percentage of jobs will have lower wait time than one of the values from the pool of measured wait time samples. Prediction accuracy of the binomial method in QBETS has been shown to be quite good (close to the actual time on average). We use a confidence level of 95%, and we find, for each application, the percentage of time that LLNL clusters are expected to have lower turnaround time than EC2 at 1024 tasks. The results are shown in Table 4 (page eight). In the table, for the 0% case, such as Sweep3D, the applications run faster on EC2 than the other clusters. For other cases, the table shows that

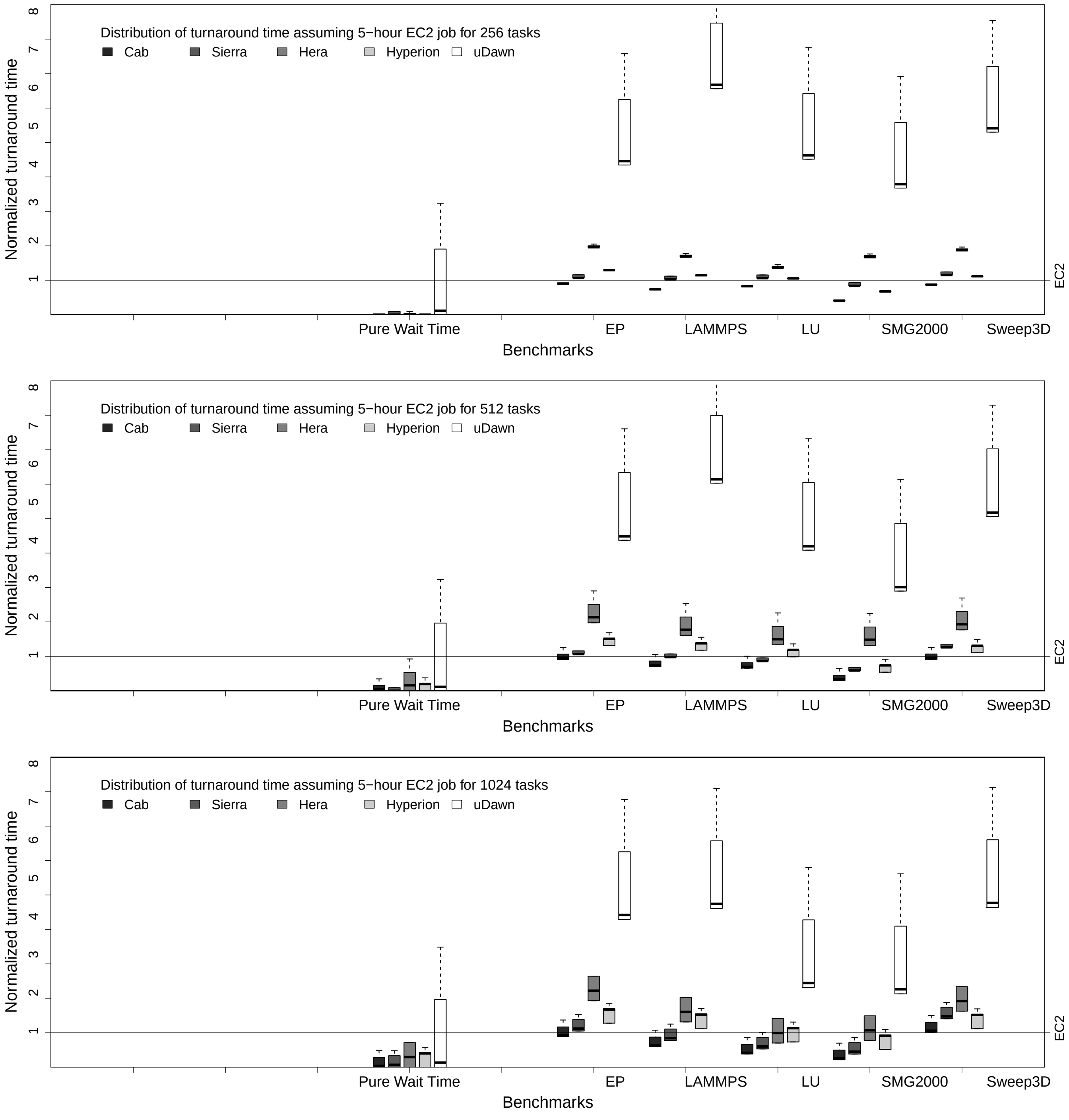


Figure 3: Comparison of total turnaround times on EC2 and LLNL clusters on 256, 512, and 1024 tasks. The figure shows turnaround times on LLNL clusters normalized to the EC2 turnaround times assuming 5-hour jobs on EC2. The y-axis represents multiples of EC2 turnaround time.

Application	Cab	Sierra	Hyperion	Hera	uDawn
EP	25%	0%	0%	0%	0%
Sweep3D	0%	0%	0%	0%	0%
LU	40%	35%	15%	25%	0%
SMG2000	35%	0%	0%	0%	0%
LAMMPS	35%	25%	0%	0%	0%

Table 4: Percentage of time that LLNL clusters are expected to have lower turnaround time than EC2 for a 1024 tasks job.

even on the fastest machine, Cab, the expectation ranges from 25% to 40%. This shows that the queue wait time can be a dominating factor on many clusters.

6. COST COMPARISON

In this section, we compare the EC2 and HPC clusters based on total cost of execution. Because clusters at LLNL are free to end users, we derive node-hour prices relative the market price of the EC2 CC2 cluster ($2.40 per node hour). We expect that clusters similar to the HPC clusters we are using in this paper will be available shortly (but we have no pricing information yet) [32]. Other factors in influencing node-hour prices include system performance, scalability and availability of resources.

This section is organized as follows. First, we present our methodology, which draws on economic theory, to price LLNL clusters. Second, using the generated node-hour prices, we present a cost-performance analysis.

6.1 Pricing Methodology

To compare clusters on the cost axis, we need to generate reasonable prices for currently un-priced clusters. One approach might be to view computation as a public utility and base prices only on the cost of ownership for the cluster operator. Cost data is not publicly available, however, and would not be publishable if we were able to obtain it; this necessitates an alternate model.

The approach we take is to assume the LLNL clusters are operated as a competitor in a market for computation. This results in generated prices that are naturally comparable to Amazon's EC2 cluster and provides for the future possibility of additional types of analysis (e.g. optimizing future machine acquisition based on the market value of individual components' effects on performance). To be clear, we are *not* implying that publicly procured computational resources should be priced this way (or priced at all). Instead, we are studying how a competitively priced situation would inform a cost-time performance analysis.

Our competitive model is developed with economic theory by nesting a model of users making optimal execution choices, given pricing and timing information, inside a model of a profit-maximizing cluster operator. Amazon's EC2 cluster is included as an outside option with a fixed price, and the operator chooses prices relative to Amazon's.

Users are assumed to have a fixed problem size. That is, they are not choosing the size of the problem to execute, only which cluster and how many nodes to split the work over. There are four user types, defined by the type of application they are running, each modeled by the speedup properties of LU, CG, BT and SP, where we took care to balance computation and communication so as not to bias the HPC or EC2 clusters. As explained later, our user model requires that we develop a continuous performance prediction function, and we use linear regression to do this. On HPC clusters, we use job scales of up to 2048 tasks to obtain sufficient data for regression, while on on EC2 we had to limit ourselves to job scales of up to 128 tasks due to limited funds. To gather enough data points to obtain sufficient significance in the linear regression, we run benchmarks 8 times at each task count. We adjust input sizes by modifying the `npb-params.h` file so that each benchmark has a total execution time of 30-500 seconds, which is large enough to compensate for variance introduced by system noise and virtualization overhead. For each cluster/application pair, execution time is modeled by a execution time function derived using a standard Ordinary Least Squares regression. Using a continuous timing model allows the user model to optimize over all possible choices of node counts, instead of restricting choices only to the set of benchmark experimental runs.

The user must choose among the available computational clusters, each with hourly price p, then decide how many nodes n to purchase. The optimal choice is defined to be that which minimizes the combined implicit and explicit costs of execution [19], $C = (p \times n \times t(n)) + (a \times t(n))$. The explicit cost, $p \times n \times t(n)$, is the actual expenditure incurred by purchasing the nodes for the time required to execute the application. The implicit cost, $a \times t(n)$, may have different interpretations depending on the context. In a business or scientific environment, this may be a literal cost incurred by waiting longer for execution to finish, for example, from giving to delay a cost-saving decision dependent on the results. In this case, parameter a is the hourly cost of waiting. Alternatively, for an individual user, a would represent the individual's personal relative valuation of time. In this case, a large a may be due to a looming deadline, or other behavioral influences that cause a person to prefer shorter execution times.

A cluster operator can then use this model of user choice to predict purchase decisions, given a candidate set of hourly node prices for each cluster. These predictions can in turn be used to achieve a particular operational objective. Here, we assume that the operator wishes to maximize profit, though achieving target utilization rates for each cluster might be a plausible alternative.

To predict the user's choice, first, for each (cluster i, user type u) pair and candidate set of prices, we solve for the optimal (cost-minimizing) number of nodes $n_{ui}(p)$. This function (evaluated for all possible prices) represents the user's classical demand function for nodes on this cluster. In our case, the functional form of $t(n)$ means that there is no closed-form solution for $n_{ui}(p)$, so n must be found with an optimization algorithm.

Next, for each user type, we calculate the probability that the user will choose each cluster using the Logit Random Utility model [28]. In short, this model states that there are unobservable factors that cause some noise to be added to the user's cost function C, and models this noise with a Logit distribution. This results in the following formula:

$$s_{ui} = e^{-C_{ui}} / \sum_j (e^{-C_{uj}})$$

s_{ui} can be interpreted in two ways: for an individual user, it is the probability that a user of type u chooses cluster i; for a population of users, it is the market share for cluster

CC2	Hera	Sierra	Hyperion	uDawn	Cab
2.40	2.36	3.83	2.13	0.25	5.49

Table 5: Node-hour prices (in dollars) for LLNL and EC2 clusters

i among users of type u. That is, it is the proportion of users of type u that choose cluster i. The operator can then choose the set of prices that results in the desired objective. Here we assume that (1) the operator is profit-maximizing over the short run, such that only hourly operational costs, and not acquisition costs, are relevant, and (2) the hourly costs of operation are effectively the same for all clusters. These assumptions mean that profit maximization is equivalent to revenue maximization. Clearly, these assumptions may not hold in the real world, but they do not substantively change the way the model works and are therefore acceptable abstractions given the absence of available cost data. Using this model, the expected profit/revenue is then calculated as the sum over the clusters and users of the expected revenue, or:

$$R = \sum_i (\sum_u (s_{ui} \times p \times n_{ui}(p) \times t(n_{ui}(p))))$$

Table 5 shows the results obtained by our model by choosing prices p to maximize R. At each iteration of the maximization algorithm, user choice as a function of price $n_{ui}(p)$ is obtained by the user's cost-minimization problem as described above.

6.2 Cost vs Performance Comparison

In this section we present cost-performance trade-offs for our MPI benchmark set using the prices obtained in the previous section. We consider pure execution times for comparison (that is, zero wait times), as we assume equal distribution of jobs across all clusters. The purpose of this section is to answer the following questions. First, what is the trade-off between execution time and cost at various scales? Second, if the user has a turnaround time bound (e.g., "finish the weather prediction for tomorrow before the evening newscast at 7pm"), is the most cost-effective way to do that always to use the fastest cluster (Cab), or might using a less powerful cluster be better?

Figure 4 shows the trade-off between cost and turnaround time. The figure is displayed as a scatterplot of turnaround time (x-axis) and cost (y-axis), with the points representing the same cluster connected to show scalability of each cluster. In general, computational scientists execute large programs for large amounts of time. For practical reasons (again, our cost on EC2), we execute short programs. Because this does not map well to the hour billing granularity used by EC2, we compensate by pro-rating the hourly rate for the given execution time. The pro-rated hourly rate is obtained by dividing the per-hour node price by 3600 (seconds per hour) and multiplying the result by the execution time. (Essentially, we are assuming the billing function is continuous instead of discrete.) In Figure 4, points closer to origin show superior configuration choice in terms of both cost and performance over other configurations. Points higher in the plot imply higher total cost of execution, while the points towards the right indicate higher execution time. Plots with smaller slopes indicate good application scalability where as larger slopes indicate poor scalability. If addition of nodes to a configuration decreases the slope, it shows better scalability, and hence adding more nodes to the configuration is cost-effective.

There are several interesting cases. First, for different applications, different clusters minimize total cost of execution. For example, the total cost of execution of running EP, Sweep3D, LAMMPS and LU is least on EC2. Except with EP, EC2 is cheapest at 256 tasks. This is because the price of EC2 is optimized for computation bound applications. This is confirmed by the nature of EC2 plot from EP to LU, with slope decreasing in proportion of computation in overall application execution. On the other hand, it is cheapest to run SMG2000 on Cab, even though the Cab nodes are most expensive. This is because the node-hour price of Cab is optimized for applications with significant communication overhead, which characterizes SMG2000 (recall Figure 1).

Second, different clusters optimize performance for certain classes of application characteristics. For example, SMG2000, EP, LU and LAMMPS are fastest on Cab, whereas Sweep3D performs best on EC2. The application characteristics for which the individual clusters are optimized include communication overhead, communication topology and cache pressure. From the architectural specifications, we can see that clusters with fast interconnects, certain cores per node and larger L2 cache per processor show better performance with applications characterized to use these resources. For example, Sierra and Cab are optimized for communication performance (with Infiniband QDR), which is well suited for SMG2000 and LU. Also, Cab, Hera, EC2 and Hyperion have cores per node in powers of two, which provides optimal communication topology for BT benchmark. Finally, EC2 and Cab are optimized for applications with high cache pressure, such as CG, as Cab has 20 MB of cache per processor.

Third, if a turnaround time bound exists, then optimizing cost may require using one of the less-powerful or more expensive clusters. For example, on LU with a turnaround time bound of 90 seconds (shown with a grey vertical line), Hera is the cheapest option at 1024 tasks. Recall that EC2, which runs more cores per node, is the cheapest option without a time bound at 256 tasks. Likewise, with a turnaround time bound of 8 seconds with EP, the cheapest choice of cluster is Cab at 1024 tasks, because both cost about the same. Without a time bound, EC2 is the cheapest option at 512 tasks. A similar situation can occur if a cost bound exists. For example, for a cost bound of $5000 per hour ($1.39 per second) with LAMMPS, EC2 is the fastest choice at 1024 tasks. Similarly, with a cost bound of $12000 per hour ($3.33 per second) with LU, Hera is the fastest choice at 1024 tasks. Without a cost bound, LAMMPS and LU run fastest on Cab at 1024 tasks. Thus, for either a time or a cost bound, we observe that the optimal choice of cluster is non-trivial.

7. DISCUSSION AND FUTURE WORK

Our evaluation reveals several interesting items. First, we establish that the choice of which cluster to use is dependent on the application, queue wait time, price, and the user's bias towards either turnaround time or total cost. Obviously, the user does not have the luxury of exhaustively trying all clusters and then deciding which was best after the fact. In our opinion, this motivates designing software

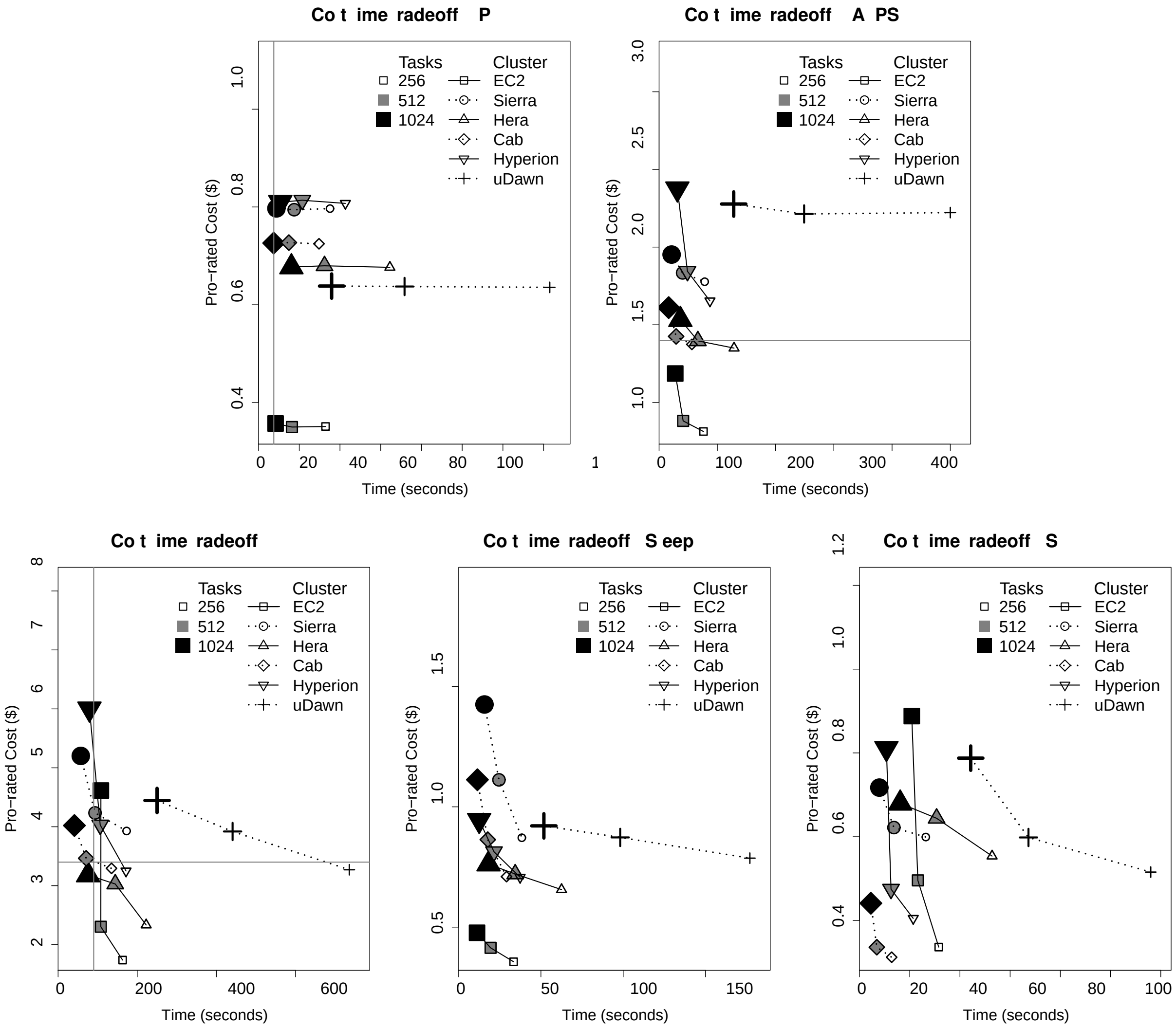


Figure 4: Cost versus turnaround time comparison for LLNL and CC2 clusters. Increasing sizes and darker shades indicate higher task counts. Different scales are used for each plot for readability. Vertical and horizontal lines indicate user cost and turnaround time constraints.

systems that perform cluster selection automatically, which has also been discussed by Li et al. [22]. While this is certainly not a simple task, an effective software system could save users a significant amount of time and money.

Second, in order to develop systems that decide between HPC clusters such as those at LLNL and EC2, it is necessary for the HPC systems to provide wait queue data—likely both historical and current. This would allow analysis to determine the expected wait time on the HPC clusters, which is clearly a critical factor in which cluster to choose. However, there are clear security concerns. These can probably be alleviated by anonymizing some queue data. On the positive side, there should be an incentive for organizations like LLNL to provide this data, as it could reduce demand.

Finally, our pricing model could be useful to the cloud provider in various ways. For example, the user's time preference parameter could be adjusted to improve prices over time depending upon real-world demands. This could be extended to obtain prices at different times of day, week or month, considering that workloads on the cloud spike up due to external known or unknown events. (For example, cloud providers could provision high-throughput resources by asking the question, "What is the time preference on the Black Friday sale?") Conversely, the pricing model could be used to incentivize use of certain resources and avoid oversubscription of other resources. The model could also indicate, based on real-world workload type and demand, which resources to upgrade. For example, if the workload consists primarily of communication-intensive applications, the pricing model would suggest an upgrade of interconnect technology. This would be indicated by low count on optimal number of nodes suggested by the pricing model.

8. RELATED WORK

There is a large body of work related to this paper. We focus on two areas in public clouds (e.g., Amazon EC2 [6], FutureGrid [5], and OpenCirrus [9]): (1) performance analysis of standard HPC benchmarks and comparative usability and (2) cost analysis of running real scientific codes on small, medium and large scale HPC-style clusters.

Amazon EC2 has become increasingly popular with scientific HPC users due to high availability of computational resources at large scale. Several researchers have benchmarked EC2 using MPI programs. Previous work [**?**, 15, 26, 16, 18, 12, 11, 13, **?**] has focused on extensively benchmarking currently available EC2 cluster types with standard MPI benchmarking suites such as NAS [7] and Sequoia [4]. Our work uses both large task counts and takes a user perspective, which has not been studied simultaneously. Also, we investigate the cost/performance tradeoff at different scales on EC2, which to our knowledge has not been investigated.

Several attempts have been made to formalize and compare the cost of running standard HPC benchmarks as well as real applications on Amazon EC2 and standard cluster systems. Formalizing the cost of a standard HPC cluster is not straightforward due to the manner in which the computational resources are charged per user. Walker et al. [30] attempt to formalize the cost of leasing CPU in HPC clusters. Work on comparing the cost of resources on medium-scale university-owned cluster with Amazon EC2 Cluster Compute (CC) instance has been carried out [10]. Cost estimation of a large-scale cluster presented by Yelick et al. [33] involves a detailed modeling of cost of ownership, support, and hardware and software upgrades. The work showed that other factors in total cost include amortized cost of a cluster, utilization rate and job execution times and input sizes. Because resources are charged on an hourly basis, attempts have been made to execute applications cost-effectively. Li et al. [22] present a comparative study of public cloud providers for different real HPC applications. Again, our work differs in the use of turnaround time and cost/performance analysis at scale.

The work most closely related to our work compares the cost of renting virtual machines in the cloud against hosting a cluster [35]. The authors present a detailed analysis of MPI applications on CC1. Also, a cost comparison between CC1 and an HPC cluster is presented with amortized cost calculations.

Our work differs from that above in several ways. Most importantly, our work studies turnaround time and cost; i.e., the perspective of the user, as opposed to the cost of running a supercomputer center. From the point of view of the owner of the center, operating an HPC cluster is always better as long as the system is reasonably well utilized, and supercomputer centers easily fit that characteristic, as they tend to be oversubscribed.

Other differences also exist with our work. First, the scale at which benchmarks were studied is typically quite small in number of cores/nodes and problem sizes. Second, most of the work employed small and medium instance types provided by Amazon EC2 that are not specifically intended for HPC applications. We present benchmarking results on the recently introduced CC2. Third, most conclusions present network latency and bandwidth, and virtualization overhead as the factors causing application performance degradation. We show that system noise is not significant compared to HPC clusters, so long as core 0 is not utilized.

9. CONCLUSION

This paper evaluated the cloud against traditional high-performance clusters along two axes–turnaround time and cost. We first confirmed prior results that high-end traditional HPC clusters are superior to the cloud in raw performance. However, we also found that queue wait times can potentially increase total turnaround times. Finally, we developed a pricing model for HPC clusters that are currently unpriced. We used the pricing model as a tool to make a fair comparison of traditional HPC and cloud resources in terms of total cost of execution. Our pricing model can be used from the perspective of the user and the provider; in this paper we focused on the user perspective. From that perspective, we found that there are multiple considerations in choosing a cluster, including the expected queue wait time along with the actual cost.

Based on this evaluation, we believe that choosing the optimal cluster is a task that should be abstracted from the typical user. Our goal in our future work is to utilize turnaround time and cost to develop tools and techniques for directing users of diverse sets of applications, given particular constraints, to the most appropriate cluster.

Acknowledgments

Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. We also thank Amazon for a grant for time on EC2. Finally, we thank our shepherd, Thilo Kielmann, as well as the anonymous reviewers for comments that improved the quality of this paper.

10. REFERENCES

[1] Parallel workloads archive. `http://www.cs.h-uji.ac.il/labs/parallel/workload/`.

[2] Simple linux utility for resource management, faq. `https://computing.llnl.gov/linux/slurm/faq.html#backfill`.

[3] ASC purple benchmarks. `https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/`, 2002.

[4] ASC sequoia benchmarks. `http://asc.llnl.gov/sequoia/benchmarks/`, 2009.

[5] Futuregrid project. `https://portal.futuregrid.org/`, 2009.

[6] Amazon. Amazon web service elastic compute cloud (EC2). `http://aws.amazon.com/ec2`.

[7] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS parallel benchmarks–summary and preliminary results. In *Supercomputing*, Nov. 1991.

[8] J. Brodkin. $1,279-per-hour, 30,000 core cluster built on Amazon EC2 cloud. `http://arstechnica.com/business/news/2011/09/30000-core-cluster-built-on-amazon-ec2-cloud.ars`, 2011.

[9] R. Campbell, I. Gupta, M. Heath, S. Y. Ko, M. Kozuch, M. Kunze, T. Kwan, K. Lai, H. Y. Lee, M. Lyons, D. Milojicic, D. O'Hallaron, and Y. C. Soh.

Open cirrus cloud computing testbed: federated data centers for open source systems and services research. In *Hot Topics in Cloud Computing*, 2009.

[10] A. G. Carlyle, S. L. Harrell, and P. M. Smith. Cost-effective HPC: The community or the cloud? In *IEEE International Conference on Cloud Computing Technology and Science*, 2010.

[11] J. Ekanayake and G. Fox. High performance parallel computing with clouds and cloud technologies. In *Cloud Computing*, pages 20–38. 2010.

[12] Y. El-Khamra, H. Kim, S. Jha, and M. Parashar. Exploring the performance fluctuations of HPC workloads on clouds. In *IEEE CloudCom*, Nov. 2010.

[13] M. Fenn, J. Holmes, and J. Nucciarone. A performance and cost analysis of the amazon elastic compute cluster compute instance. `http://rcc.its.psu.edu/education/white_papers/cloud_report.pdf`, 2011.

[14] Y. Gong, B. He, and J. Zhong. An overview of CMPI: network performance aware MPI in the cloud. In *ACM PPOPP*, Feb 2012.

[15] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn. Case study for running HPC applications in public clouds. In *ACM HPDC*, 2010.

[16] Z. Hill and M. Humphrey. A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: the death of the local cluster? In *International Conf. on Grid Computing*, Oct. 2009.

[17] K. Z. Ibrahim, S. Hofmeyr, and C. Iancu. Characterizing the performance of parallel applications on multi-socket virtual machines. In *IEEE/ACM CCGrid*, 2011.

[18] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance analysis of high performance computing applications on the Amazon web services cloud. In *IEEE CloudCom*, Nov. 2010.

[19] G. A. Jehle and P. J. Reny. *Advanced Microeconomic Theory*. Prentice Hall, 2000.

[20] D. Klusáček and H. Rudová. Alea 2 – job scheduling simulator. In *SIMUTools*, 2010.

[21] S. H. Langer, B. Still, P.-T. Bremer, D. Hinkel, B. Langdon, J. Leviney, and E. Williams. Cielo full-system simulations of multi-beam laser-plasma interaction in nif experiments. In *Cray Users Group Meeting*, May 2011.

[22] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: comparing public cloud providers. In *IEEE Conference on Internet Measurement*, pages 1–14, 2010.

[23] J. D. McCalpin. The STREAM benchmark. `http://www.cs.virginia.edu/~mccalpin/STREAM_Benchmark_2005-01-25.pdf`.

[24] D. Nurmi, J. Brevik, and R. Wolski. Qbets: Queue bounds estimation from time series. In *Wkshp on Job Scheduling Strategies for Parallel Processing*, Jun 2007.

[25] F. Petrini, D. J. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Supercomputing*, 2003.

[26] F. Schatz, S. Koschnicke, N. Paulsen, C. Starke, and M. Schimmler. MPI performance analysis of Amazon EC2 cloud services for high performance computing. In *Advances in Computing and Communications*, pages 371–381. 2011.

[27] D. Singh. personal communication, Mar. 2012.

[28] K. E. Train. *Discrete Choice Methods with Simulation*. Cambridge University Press, 2009.

[29] E. Walker. Benchmarking Amazon EC2 for high-performance scientific computing. *Login*, 33(5):18–23, Oct. 2008.

[30] E. Walker. The real cost of a CPU hour. pages 35–41, 2009.

[31] Wikipedia. Cloud computing wikipedia page. `http://en.wikipedia.org/wiki/Cloud_computing`, 2011.

[32] Windows Azure Big Compute. `http://www.windowsazure.com/en-us/home/features/big-compute/`.

[33] K. Yelick, S. Coghlan, B. Draney, and R. S. Canon. The magellan report on cloud computing for science. `science.energy.gov/~/media/ascr/pdf/program-documents/docs/Magellan_Final_Report.pdf`, December 2011.

[34] L. Youseff, R. Wolski, B. Gorda, and C. Krintz. Evaluating the performance impact of xen on MPI and process execution for HPC systems. In *International Workshop on Virtualization Technology in Distributed Computing*, 2006.

[35] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen. Cloud versus in-house cluster: evaluating amazon cluster compute instances for running MPI applications. In *Supercomputing*, Nov. 2011.