



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

A User Perspective of High-Performance Computing on the Cloud

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

May 7, 2012

SC2012
Salt Lake City, UT, United States
November 10, 2012 through November 16, 2012

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 User Perspective of High-Performance Computing on the Cloud

Aniruddha Marathe*, David K. Lowenthal*, Barry Rountree†, Xin Yuan‡, Martin Schulz†, Bronis de Supinski†

*Department of Computer Science, The University of Arizona

†Lawrence Livermore National Laboratory

‡Department of Computer Science, Florida State University

Abstract—Cloud computing has increased greatly in popularity in recent years. However, many HPC users do not believe that the cloud is a viable alternative for executing all but embarrassingly parallel scientific applications. In this paper, we show that, while by execution time the cloud is sometimes not a viable alternative, *this view is narrow*. A more general view has to consider, from the user’s point of view, *turnaround time and cost*.

We evaluate raw performance of HPC applications on the top-of-the-line cluster offered by Amazon EC2, and compare the results to high-end clusters at Lawrence Livermore National Laboratory (LLNL). We confirm prior results that the performance gap between EC2 and high-end clusters stems primarily from limited bandwidth. However, we find, interestingly, that (1) EC2 can be better than using the LLNL clusters when considering turnaround time and (2) the cost effectiveness of different EC2 clusters varies based on the application.

I. INTRODUCTION

In recent years, the cloud has had significant success in the commercial arena, but the story for high-performance computing (HPC) has been rather disappointing. 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 [7], [8]. A more complicated question is: how well does the cloud do on more tightly-coupled applications? The prevailing opinion is that the cloud is more-or-less useless for such applications [19], [29].

Many reasons justify skepticism of the cloud for tightly-coupled, more traditional HPC applications. First, the latency and bandwidth of the network used by the cloud is usually inferior to that of a traditional cluster (e.g., Ethernet vs. Infiniband). Second, cloud nodes are virtualized, which causes concerns in terms of virtualization overhead as well as virtual machine co-location. Finally, system noise when executing on the cloud, which may be a direct result of interference from node virtualization, is a concern.

However, to compare the cloud, which provides a fee-for-service model in which access is essentially available 24/7, to traditional HPC clusters on only the axis of traditional 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 (they are “free” to the user). These properties hold not only for the Lawrence Livermore National Laboratory (LLNL) on which we experiment but also for HPC

clusters hosted, for example, at a company or university as well as government laboratories.

We evaluate the cloud against traditional HPC clusters but with a more generalized evaluation scheme. We compare the Amazon EC2 cloud to two LLNL HPC clusters for a typical set of HPC benchmarks. Our work differs from prior work, such as Zhai et al. [32], in that we evaluate not just execution time, but also total turnaround time and cost; i.e., the factors of interest to the user. We also compare the various cloud instances—specifically, Cluster Compute Quadruple Extra Large, Cluster Compute Eight Extra Large, and High-CPU—against each other and argue that applications should be mapped to the most appropriate cluster, which is not necessarily the highest performing one.

The contributions of this paper are:

- An evaluation of EC2 and HPC clusters along the axis of execution time at reasonable scales (over 1000 cores).
- An evaluation of EC2 and HPC clusters along more general axes, including total turnaround time and cost, providing, for the first time to the best of our knowledge, a *user perspective* of HPC on the cloud (prior work studies the data center perspective [29], [32]).
- An execution time and cost comparison of EC2 instances.

Our results show that the choice of EC2 versus HPC clusters is more complicated and nuanced than previously thought. First, EC2 nodes are high end and, thus, execution time is better on EC2 for some applications that incur modest communication. For example, at 1024 tasks, the top EC2 cluster executed LAMMPS 21% faster than the top-end LLNL cluster. However, we confirm prior results showing that communication intensive applications are typically inefficient on EC2 [29].

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 time. For example, when the wait time exceeds its median, total turnaround time on the LLNL machines is often larger than that on EC2 (by more than a factor of 2) even though the application may execute several times faster.

Finally, we show that it may cost less to use an inferior EC2 cluster. For example, using the least-cost EC2 cluster with certain time bounds is nearly 10% less expensive than the highest-cost, most capable EC2 cluster. Further, this difference

can increase as the number of tasks scales or if the EC2 price structure more closely matched cluster quality.

The rest of this paper is organized as follows. Section II provides background of EC2 and motivates our comparison of EC2 to HPC clusters. Section III describes the machines, provides our experimental setup, and discusses how we derive wait queue times. Section IV contains our evaluation, and Section V discusses the implication of our results. We provide related work and our conclusions in Sections VI and VII.

II. 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 [28]. 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.

A. EC2 Basics

We focus on the most popular cloud platform, which is Amazon EC2 [5]. 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 the host operating system (often called the hypervisor). There can be many virtual machines in one physical machine, and so 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 three “regular” instances (small, large, and extra large), one micro instance, three “high memory” instances (extra large, double extra large, and quadruple extra large), two “high CPU” instances (medium and extra large), and three “cluster compute” instances (one quadruple extra large, one eight extra large, and one with a GPU). These instances differ in their computational and network capabilities.

Amazon EC2 also markets several kinds of ways of purchasing time on their systems. One way is called “on-demand”, in which the user pays money for each VM instance and receives access to the purchased node immediately. Another option is called “reserved”, in which the user pays a yearly fee, but then when the user purchases time, the rate is lower than that of on-demand¹. Finally, there is the “spot market”, where users can bid on nodes and potentially pay much less. The spot market model is nontrivial and has been explained elsewhere [30]. In this paper we investigate only on-demand instances, while noting that the other options are quite interesting and may be superior in certain situations. For the purposes of this paper,

¹There are multiple types of reserved instances, depending on the user’s expected machine utilization over time.

the key is that on-demand instances have a given cost (see below) and zero queue wait time (unlike batch schedulers) in most circumstances².

Other researchers have found (in the past) that EC2 does not provide any locality guarantees, which is often an important property to achieve efficient performance for HPC applications. In particular, if a user of EC2 requests N nodes to create a cluster, there is in general no guarantee of which N nodes the user receives [15]. This can be critically important in tightly-coupled HPC applications, and it is why batch systems such as Moab [4] may increase queue wait time in order to return a more desirable set of nodes. However, EC2 does allow this sort of physical proximity through a *placement group* on their highest-end clusters (the Cluster Compute family). However, it is not clear how this is implemented (in other words, it is an open question how many nodes in a placement group a user can acquire without wait times similar to batch systems). In our experiments, we did not have any delay due to placement groups, but we did not use more than 128 nodes. For most systems, batch systems cannot guarantee physically proximate nodes either (they perform best effort) [20].

B. Comparing EC2 to HPC Clusters

This paper compares EC2 clusters with two clusters that reside at Lawrence Livermore National Laboratory (LLNL): Hera and Sierra (discussed more in the next section). These are typical high-end clusters that use Infiniband networking. They serve as an example of what we denote “HPC clusters”, that are owned and operated by an organization to solve key problems for individuals in the organization and their collaborators. Generally speaking, the clusters are well utilized (otherwise they would not be cost effective). We will use the terms “LLNL clusters” and “HPC clusters” interchangeably in the rest of this paper.

Considering that the HPC clusters are assumed here to be “free” and have better network infrastructure, one may ask why one would ever execute an HPC application on a cloud cluster as opposed to an HPC cluster (which, again, we assume runs a batch scheduler).

- 1) The application may be compute intensive, and some of the nodes offered by EC2 may execute such applications faster than HPC nodes, since cloud providers typically can afford a faster upgrade/refresh cycle for their machine park.
- 2) 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.
- 3) A given user may not have access to an HPC cluster (e.g., security issues at a national laboratory). We do not consider this further in the paper, as it is fairly obvious if one has access only to EC2, then there is nothing to study aside from comparing different EC2 clusters.

²Of course, there are a finite number of EC2 machines, so there exists a number of nodes that would cause the user to wait until more capacity exists.

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”. There obviously is some operational cost that could be passed on to the user, but it is certainly much less than what Amazon, as a for-profit company, charges.

Thus, there is a tradeoff 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 of this paper is to try to characterize these factors and to better understand in which situations using EC2 for traditional HPC apps makes sense.

We do not make any judgments about the relative importance of turnaround time and cost. Such an ordering depends on the relative importance of these quantities to the user. It is also possible that the user operates under a time (or cost) bound, and wishes to minimize the cost (or time) such that the constraint is honored. In this paper, we will merely present and analyze data, but our eventual goal is to be able to determine, without exhaustive execution, the best configuration (which type of machine plus number of nodes).

III. EXPERIMENTAL SETUP

This section describes our experimental setup. First, we provide a description of all test systems and benchmarks used in our evaluation. Second, we describe how we set up the experiments. Third, we discuss how we evaluate wait queue times.

A. Machine and benchmark descriptions

Table I shows configurations for our test systems. Two of our systems reside at Lawrence Livermore National Laboratory (LLNL). Sierra is one of the newer systems and has 1849 Intel Xeon 5660 nodes, 12 cores per node, clock speed of 2.8 GHz, 12 MB cache, memory size of 24 GB/node, and Infiniband QDR inter-node connectivity. Hera is a somewhat older system; it has 800 quad-core Opteron nodes, 16 cores per node, clock speed of 2.3 GHz, 512 KB cache, memory size of 32 GB/node, and Infiniband DDR inter-node connectivity.

We used three different EC2 clusters: Cluster Compute Quadruple Extra Large “CC1”, Cluster Compute Eight Extra Large “CC2”, and High-CPU Extra Large (“HC”). A CC1 node is a Xeon X5570 processor, with two quad-cores, clock speed of 2.93 GHz, 8 MB cache, memory size of 23 GB/node, and 10 Gb Ethernet inter-node connectivity. A CC2 node is a Xeon Sandy Bridge processor, with two oct-cores, clock speed of 2.59 GHz, 20 MB cache, memory size of 60.5 GB/node, and 10 Gb Ethernet inter-node connectivity. An HC node is a Xeon E5410 processor, with two quad-cores, clock speed of 2.33 GHz, 6 MB cache, memory size of GB/node, and 1 Gb Ethernet inter-node connectivity. Currently, Amazon has two kinds of HC nodes and returns them arbitrarily; the other type is also a Xeon but 2.13 GHz. Because we assume

homogeneous computing in this paper, we discarded any 2.13 GHz node. Obviously, this incurs a cost (Amazon has no return policy), but for long-running computations, this one-time cost will be amortized; so, we do not count the cost of discarding in our results.

The relative computational power of CC1 and CC2 (according to Amazon), relative to HC, is 1.68 and 4.40, respectively. The costs per hour for CC1, CC2, and HC, respectively, are \$1.30, \$2.40, and \$0.66. As mentioned earlier, all EC2 clusters are virtualized, though CC1 and CC2 have hardware support to reduce overhead.

The benchmarks we used are from the NAS Parallel [6], ASC Sequoia [2], and ASC Purple [1] benchmark suites. Specifically, we ran 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. The cost per hour at scale (128 nodes/1024 tasks) on CC2 is over \$300.

For all programs, we configured the benchmarks so that they would run for between 30 and 120 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 used 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 used the `makeinput` utility and modified the sizes. For LAMMPS, we used the Lennard-Jones input deck. In this paper, we use strong scaling, so in each set of results, all of the benchmark sizes were identical across different MPI task counts.

The benchmarks we used had a variety of message characteristics. Many were communication intensive (BT, CG, LU, and SP), sending, per MPI rank, at least 100K messages totaling at least 1 GB. (SP sent over 2 GB per rank.) SMG2000 sent about 400 MB per rank. LAMMPS sent 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.

B. Program setup

We used MVAPICH-1.7 (for the LLNL clusters) and MPICH2 (for the EC2 clusters). We compiled all benchmarks using the `-O2` option. Also, all experiments use 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). This is because currently the EC2 systems pin all interrupts on to core 0. For communication intensive programs, this causes severe load imbalance on core 0 and significant performance degradation. Separate tests show as much as a 500% overhead on such programs. Personal communication with Amazon indicates that in the near future interrupt handling will be spread throughout the cores [25]. We would expect that this problem will then cease to exist.

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	—
Cluster Compute Quadruple (CC1)	2.93	8	23	8	10 GigE	1.3
Cluster Compute Eight (CC2)	2.59	20	23	16	10 GigE	2.4
High-CPU (HC)	2.33	6	23	8	1 GigE	0.66

TABLE I
SYSTEM SPECIFICATION FOR OUR TEST SYSTEMS

Performance interference arising from pinning interrupts to core 0 is a known problem, first reported by Petrini [23] and can be addressed easily by simply leaving core 0 unused. However, this 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. To level the playing field, we therefore chose to leave half the cores unused. We also disabled hyperthreading on the EC2 clusters. This is because we found that hyperthreading most often degrades performance.

C. Wait queue times

Part of the next section is concerned with comparing total turnaround times, which is the time between submission of the job and completion of the program. Sierra and Hera (and all oversubscribed machines that use batch submission) are optimized for execution time. On the other hand, EC2 optimizes for turnaround time [25].

To measure turnaround time, we need to know wait queue time. This is difficult to measure, because wait queue time is not constant. On EC2, using on-demand instances with non-excessively-sized requests, there is no wait queue time. Sierra and Hera, however, are well utilized by a large number of users and hence can have significant wait queue time. In our at-scale experiments, we always asked for 128 nodes. Because Sierra and Hera have 1849 and 800 nodes, respectively, this allowed us to examine two situations, one of which will likely have much lower wait queue time than the other. We also ran tests at 32 and 64 nodes, which on average incur less wait queue time.

Estimating wait time is particularly tricky because (1) the batch submission algorithm used is opaque, and (2) wait queue time may not be linear. To estimate the wait time, we submitted jobs to both Sierra and Hera at 10am and 10pm every day for a month. We set the maximum job time (the time at which, the program, if still executing, is killed) to 2 minutes or 5 hours. We choose the latter (somewhat arbitrarily) as a “typical” supercomputing job, and the former acts as a lower bound for queue wait time and also investigates how wait queue time varies with the requested maximum job time. The actual “job”

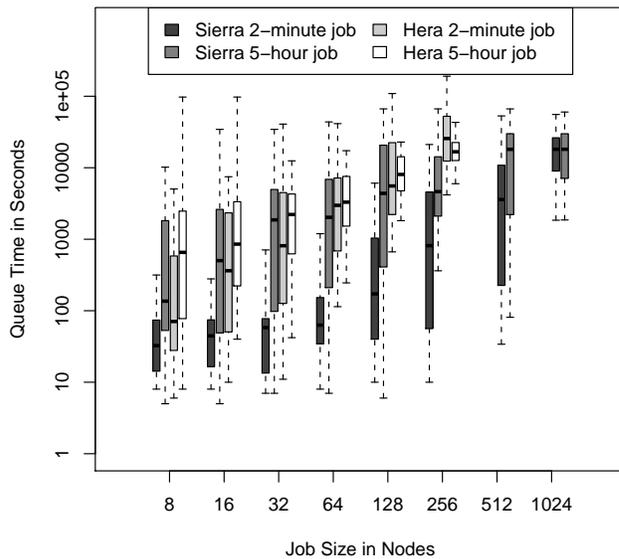


Fig. 1. Boxplot showing comparison of queue wait times at different node counts on Sierra and Hera

we ran simply recorded the time at which the “job” started and then exited. We did this for a minimum of 8 nodes and a maximum of 1024 nodes on Sierra (and 256 nodes on Hera, which is the maximum obtainable in normal operating mode).

The results are shown in Figure 1 as a series of boxplots, which shows the median in addition to the ranges of each quartile. Wait queue 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 512 or 1024 nodes on Sierra). In addition, the data shows that even for the two-minute jobs, there are potentially significant wait times.

IV. EVALUATION

This section describes our evaluation. First, we present results of microbenchmarks on EC2 and LLNL machines to evaluate their network latency and bandwidth as well as computation power. Second, we provide our results of comparing EC2 to LLNL clusters in terms of execution time, turnaround

	Hera	Sierra	CC1/CC2	HC
Latency	5.3 μ s	3.7 μ s	110 μ s	256 μ s
Bandwidth	4.1 Gb/s	6.9 Gb/s	1.4 Gb/s	0.2 Gb/s

TABLE II
NETWORK LATENCY AND BANDWIDTH FOR HERA, SIERRA, CC1, CC2,
AND HC.

	Hera	Sierra	CC1	CC2	HC
Relative performance	1.00	2.21	2.19	2.28	1.51

TABLE III
RELATIVE SEQUENTIAL PERFORMANCE (NORMALIZED TO HERA) FOR
SIERRA, CC1, CC2, AND HC.

time, and cost on the strongly-scaled benchmarks described previously with 256, 512, and 1024 MPI tasks using both execution time and total turnaround time. Third, present cost analysis across the different EC2 clusters (Cluster Compute Quadruple Extra Large [CC1], Cluster Compute Eight Extra Large [CC2], and High-CPU Extra Large [HC]).

A. Microbenchmarks

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

First, we present the results of network latency and bandwidth. We developed our own synthetic MPI benchmark that measures the round-trip time for various data sizes using a data size of 1 byte. For bandwidth tests, we used a data size of 100K. Table II shows the ranges of times collected on the test clusters. The experiments showed that Sierra has the least inter-node latency and the fastest bandwidth. This is due to the Infiniband QDR technology used for interconnection. CC1 and CC2 (which use identical networks) show low variance in network bandwidth; however, the latency is about 30 times higher than Sierra. These times are consistent with what others have found [32], [29]. The 1 Gb/s network used by HC causes its bandwidth to be lower (by a factor of about 7), and latency is likely higher than CC1/CC2 (by a little more than a factor of 2) because of the lack of guaranteed physical proximity of the allocated nodes.

Importantly, the network variance on HC was larger than that on CC1/CC2; in particular, bandwidths varied up to 50% in our experiments in HC, compared to only 10% in CC1/CC2. This is, again, likely because of the physical proximity of the nodes on CC1/CC2.

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. Also, the network overhead on EC2 machines dominates all other overheads. However, as shown earlier, sequential performance on CC1 and CC2 nodes is quite good, even if virtualization overhead exists (plus, there is hardware support to reduce it).

Table III shows the result of executing all of our benchmarks on one MPI task, so there is no communication. Each machine

is expressed in terms of average speedup over Hera, which is the slowest node. The CC2 node is the highest performing one, and likely the difference could be larger in cases where there is significant memory pressure, as the CC2 cache, at 20 MB, is much larger than any of the other nodes we used.

Also, EC2 does not co-locate virtual machines on CC1, or CC2 [25]. Finally, we have already discussed the noise issue; when omitting core 0, there is not a significant noise overhead.

B. Execution Time at Scale

In this section, we first present results of our MPI benchmarks that allow us to compare CC2, Sierra, and Hera. We use 128 nodes and a total of 1024 MPI tasks (8 cores per node). 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 2 shows the median values collected during at least three runs on the systems (normalized to CC2 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 [32], [19], in the sense that communication-intensive applications have significant overhead on the cloud. BT and CG performed significantly worse on CC2 than Sierra or Hera (nearly an order of magnitude compared to Sierra, and a factor of 3-4 on Hera), primarily due to communication overhead. LU, SP, and SMG2000 perform 3-4 times worse on CC2 than Sierra, but are generally less than a factor of 2 slower on CC2 than Hera. On EP and Sweep3d, CC2 is slightly slower than Sierra (4.6% and 9.2%, respectively), while Hera is slower by 51% and 44%.

Somewhat surprisingly, CC2 outperforms both Hera and Sierra on LAMMPS. While these two codes do have nontrivial communication at 1024 tasks, the superiority of the CC2 nodes more than compensates for the inferior communication infrastructure. Again, this is because CC2 is comprised of newer nodes than Sierra or Hera. We would expect this difference to evaporate were we to compare to the newest clusters that LLNL has acquired, which, like CC2, have Intel Xeon dual-socket, oct-core nodes.

We also note that Hera is significantly slower than Sierra, commonly running between two and three times slower. This is because Hera is a much older machine, and has much slower processors. The point is that people use Hera; as will be seen in the next subsection, it is quite oversubscribed. Thus, there is at least some reason to believe that *some* tightly-coupled parallel computing applications could be run in the Cloud, even with its inferior communication latency and bandwidth.

C. Turnaround Time

In this section, we focus on the same three machines (CC2, Sierra, and Hera), but turn our attention to total turnaround time. Figure 3 (page 7) presents a statistical representation of total turnaround time on both Sierra and Hera at three different MPI task counts: 256, 512, and 1024. (For CC2,

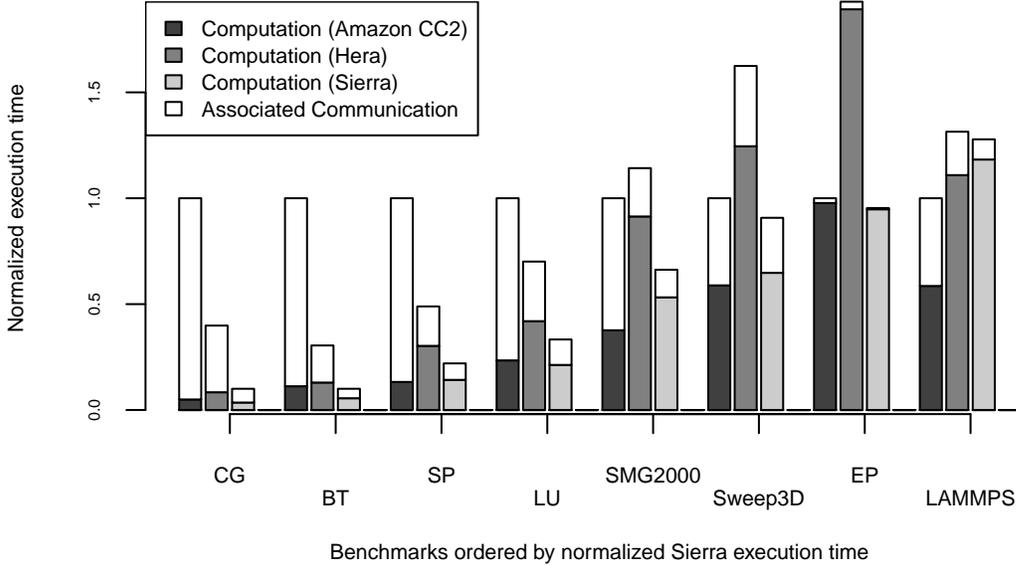


Fig. 2. Comparison of execution times on Sierra, Hera and CC2 clusters (128 nodes/1024 tasks). Times are normalized to those of CC2, and the relative percentage spent in computation and communication is shown.

total turnaround time is identical to execution time.) The data is presented as follows. We took the execution time of Sierra (Hera) and normalized it to the CC2 execution time and then placed it on a 5 hour scale. (For practical reasons, we had to run short jobs because of the overwhelming cost we would have incurred to run for 5 hours on CC2.) We then add to the 5-hour normalized execution time to the queue wait time to achieve total turnaround time. Because queue wait time is a distribution, so we use a boxplot to represent it the results we collected for the 5-hour wait times.

The results show two broad things. First, in many cases, the CC2 execution time is better at lower scales. In such situations, of course, total turnaround will be much better on CC2, as queue wait time is an additive penalty on Sierra and Hera. However, as we increase the number of MPI tasks, Sierra and Hera scale better than CC2; again, this is not surprising as (1) we are using strong scaling, and (2) Sierra and Hera use Infiniband and CC2 uses 10 Gb Ethernet.

Second, consider the most interesting comparison, which is Sierra (the more powerful LLNL cluster) and CC2 at 1024 MPI tasks. Clearly, the queue wait time governs turnaround time. That is, for all applications other than LAMMPS, if the queue wait time on Sierra falls within the first quartile, Sierra is superior (due to the difference in execution time). On the other hand, if the queue wait time falls in the fourth quartile, then CC2 is clearly better. If the queue wait time falls in the second or third quartile, which system is better depends on (1)

Application	Hera	Sierra
BT	50%	55%
CG	55%	55%
EP	—	20%
LAMMPS	—	—
LU	25%	55%
SMG2000	—	50%
SP	45%	55%
Sweep3d	—	25%

TABLE IV
PERCENTAGE OF TIME THAT HERA AND SIERRA ARE EXPECTED TO HAVE LOWER TURNAROUND TIME THAN CC2.

where in the quartile the wait time falls, along with (2) the relative superiority of the execution time on Sierra. As before, if the execution time is superior on CC2 (as for LAMMPS), then the turnaround time is always lower on CC2.

We can also view the expected wait time using existing methods such as QBETS [22]. 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 used a confidence level of 95%, and we found, for each application, the percentage of time that Hera and Sierra are expected to have lower turnaround time than CC2 at 1024 tasks. Table IV

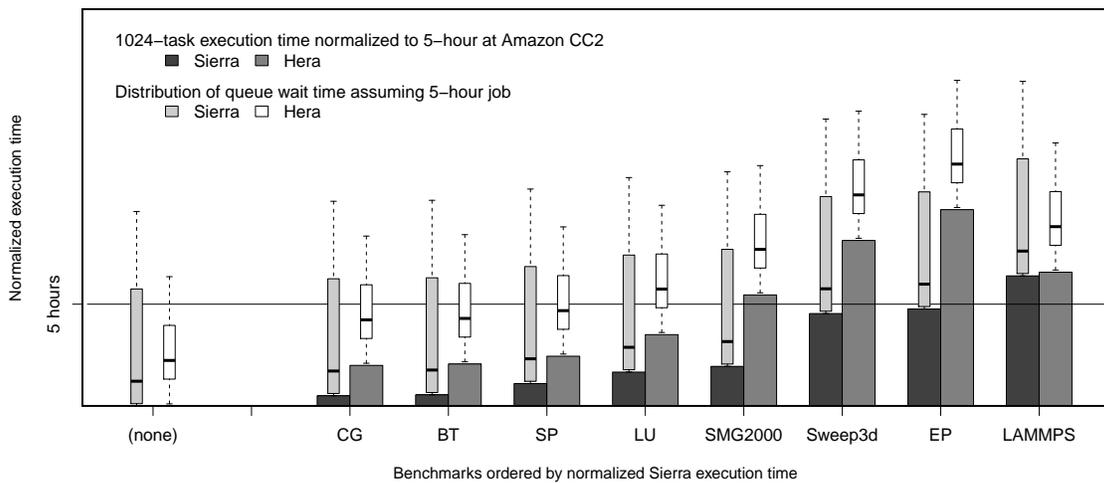
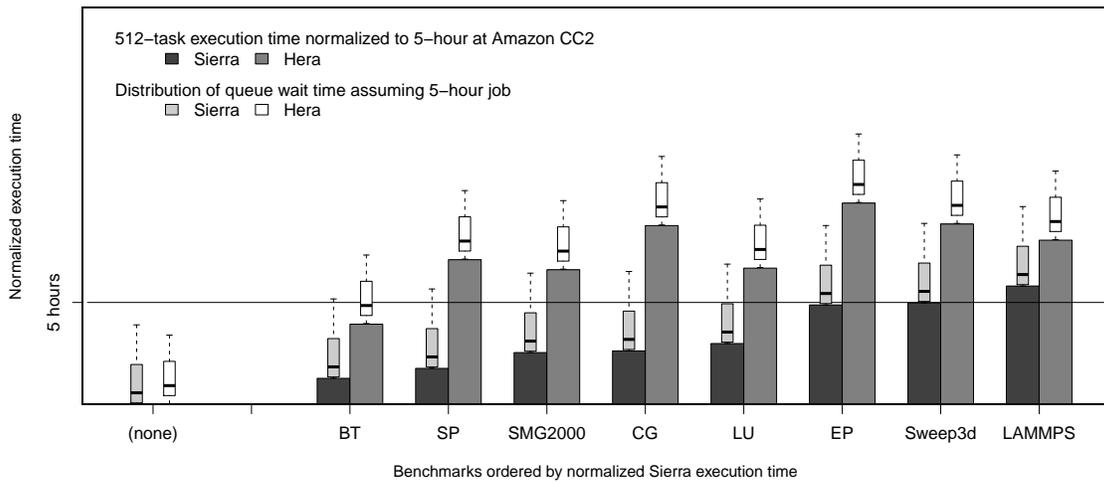
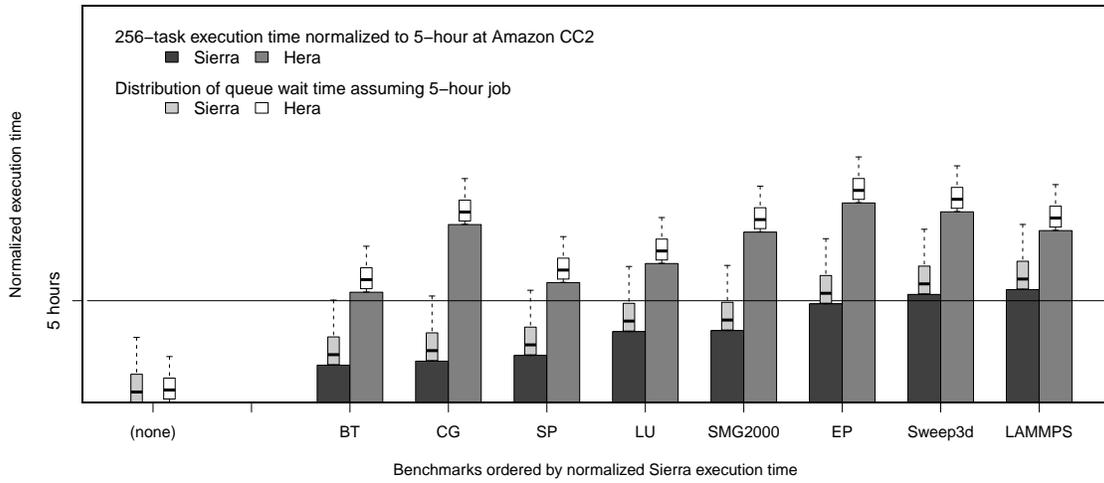


Fig. 3. Comparison of total turnaround times on Hera and Sierra on 256, 512, and 1024 tasks. The figure shows execution times, normalized to the CC2 time on a 5 hour scale, along with a boxplot of the queue wait time.

shows that on Sierra, the expectation ranges from 20% to 55%. With Hera, the expectation ranges from 25% to 55%. However, on Hera, there is no percentage to compute for EP, LAMMPS, SMG2000, and Sweep3d, as CC2 executes faster and therefore is always better in turnaround time. This only occurs with LAMMPS on Sierra.

Of course, the cost to the user of Sierra and Hera (technically, “free”) is much less than the cost of CC2. It is up to the user to determine whether the cost of renting a cluster is worth the certainty of avoiding any queue wait time. Even if we assume the user pays to use the LLNL machines at cost (others have derived these costs, per node-hour, as 3 cents on the low end [29] and 26 cents on the higher end [10], depending on the assumed utilization and other factors), the lower cost and faster communication performance mean that EC2 will always be a more expensive alternative than will a reasonably utilized HPC cluster.

The next subsection discusses incorporating cost for different EC2 clusters.

D. EC2 Cost/Performance Comparison

In this section we study the differences between three EC2 clusters: CC1, CC2, and HC, from a cost and performance perspective. While we will continue to use turnaround time as our terminology, it is the same as execution time for on-demand EC2 instances. The purpose of this section is to answer the following questions. First, what is the tradeoff between turnaround 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 CC2, or might using a less powerful cluster be better? In this section we use smaller scales (16, 32, and 64 tasks); this is because of financial reasons, but we did lower the problem sizes from the previous subsections to maintain similar computation-to-communication ratios.

It is important to note that CC2 is priced much better than CC1 or HC. Keep in mind that CC2 is nearly 3 times more powerful than CC1, yet costs less than twice as much; also, CC2 is nearly 4.5 times more powerful than HC, yet costs less than 4 times as much. In other words, users can get more for their money with CC2, even though traditionally one receives less for their money with the highest-end systems. Thus, it is entirely possible that although HC and CC1 are cheaper per-node-hour than CC2, CC2 may be both faster *and* cheaper than either—considering that users are charged for usage.

Figure 4 shows the tradeoff between cost and turnaround time. The figure is displayed as a scatterplot of cost (x-axis) and turnaround time (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 had to execute short programs. This does not map well to the hour billing granularity used by EC2. Therefore, to calculate cost, we take the hourly rate and divide

it by the actual time spent. (Essentially, we are assuming the billing function is continuous instead of discrete.)

Armed with these graphs, we can consider the question of whether the best performance and cost is universally achieved with CC2—as CC2 nodes provide the best price/rated performance. Figure 4 shows that this is not the case. In fact, the least cost is achieved with HC on four of our eight applications (EP, LAMMPS, LU, and Sweep3d, on either 16 or 64 tasks, depending on the particular application). (In the figure, we show only six of the eight graphs due to space limitations.) The lowest cost sometimes occurs on HC because despite the difference in the rated raw core-specific performance of the CC1, CC2, and HC, the actual relative difference in turnaround time is often much smaller. This can be either because the actual computation does not map to benchmarks that EC2 used to determine the performance difference between their node types, or because communication time is significant (or both). The latter point is visible primarily in comparing CC1 and CC2, as they share identical network infrastructure.

Still, CC2 is generally fastest. We next answer the question posed above about whether in cases in which a turnaround time bound exists, the lowest cost occurs with CC1 or HC. If we consider a time bound (horizontal line) in Figure 4 at various points, we see that this indeed can occur. For relative turnaround time bounds (normalized, as the graphs are, to CC2 at 64 tasks) of 1.73, and 5.95, respectively, LU (at 64 tasks) and LAMMPS (at 16 tasks) have least cost on HC (by between 1.7% and 8.5%) compared to the corresponding least costs of CC2. This difference can be larger if there is a larger variety of task counts. However, unsurprisingly, there are many cases where the clear best choice is to use CC2. An example is Sweep3d, where no matter what time bound is chosen, CC2 is the right choice (with as many tasks as needed to satisfy the time bound).

Because CC2 compared to CC1 has twice as many cores, the same network infrastructure, and costs less than twice, CC2 essentially dominates CC1. By the same token, CC2 and HC differ by approximately a factor of 5 in computation and 7 in bandwidth, yet CC2 costs 3.6 as much. If the systems had “true” prices, selecting the best cluster would be a much more complicated and interesting problem.

Still, there exists no case in which CC2 dominates; in other words, there is no single configuration (type of node and number of nodes) that is always better in turnaround time and cost. For this to happen, it would be necessary to have linear scaling *and* a performance gap between CC2 and the other two that is relatively larger than the cost difference. So, the least-cost cluster depends on the application and time bound. Obviously, the key is to determine this ahead of time as opposed to after the fact through exhaustive executions.

V. DISCUSSION

Our evaluation has revealed several interesting items. First, we have established that the choice of which cluster to use is dependent on the application, queue wait time, and price. Obviously, the user does not have the luxury of exhaustively

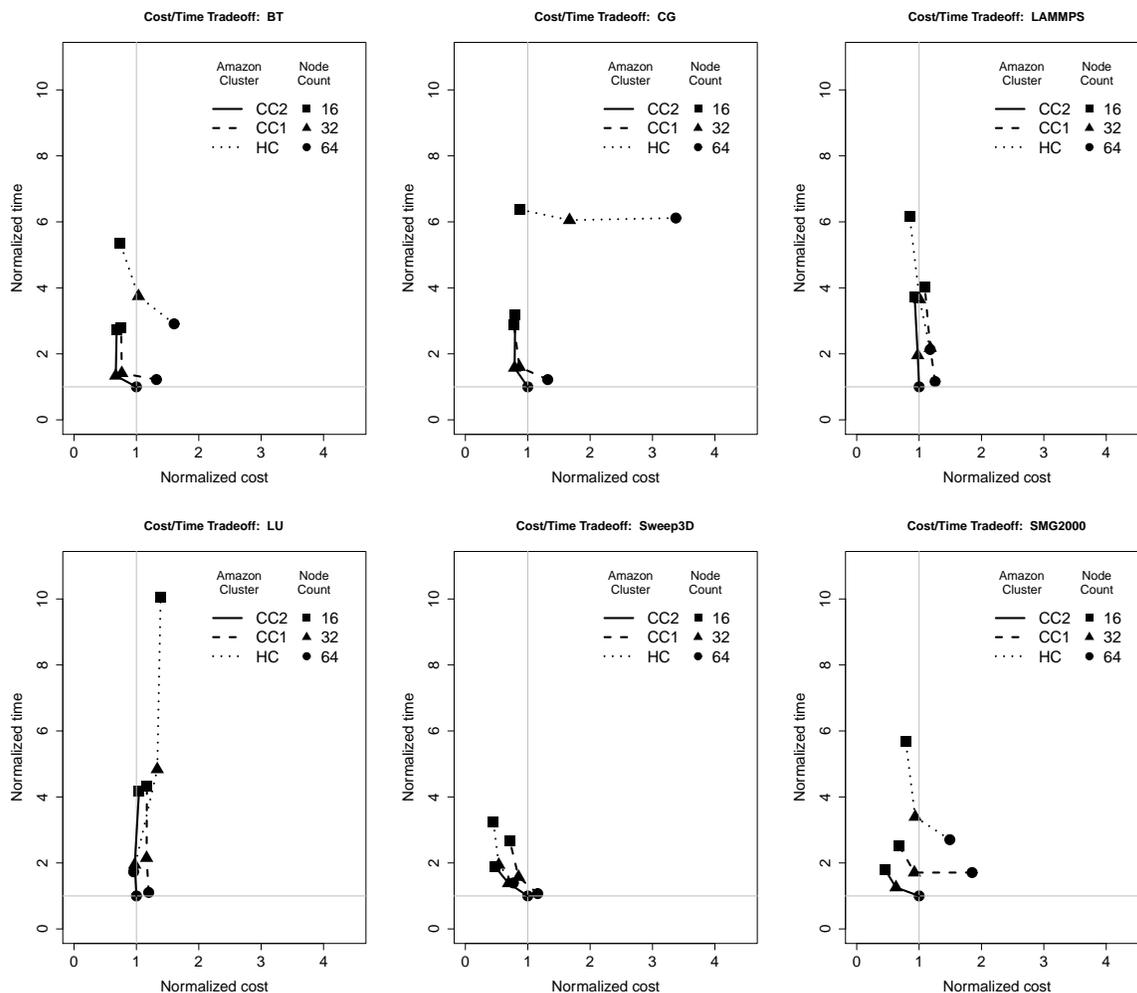


Fig. 4. Cost versus turnaround time comparison for CC1, CC2, and HC.

trying all clusters and then deciding which was best after the fact. In our opinion, this motivates designing software systems that perform cluster selection automatically, which has also been discussed by Li et al. [21]. While this is certainly not a simple task, an effective software system could save users a significant amount of money. (Also, while we did not investigate cost bounds in this paper, given such a bound, presumably a software system could save the user time.) In general, a user could, for example, be choosing between a desktop machine, a department cluster, a university-owned cluster, a national-lab-owned cluster, and the cloud (even though we studied only the last two kinds of systems). Each has its own performance and cost tradeoff.

Second, the cost-performance tradeoff seems to us fairly likely to become more interesting going forward. Currently, for example, CC2 is priced attractively, causing it to be the best option in many cases. Amazon already lowered the price of CC1 from \$1.60 to \$1.30 upon the arrival of CC2, and we would not be surprised if that price drops further with the natural migration of users to the highest-end cluster available.

Third, 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 some demand.

Finally, the sort of results shown in this paper could motivate owners of HPC clusters to use the cloud for overflow capacity. Obviously, this depends on the particular organization and the importance and timeliness of jobs.

VI. RELATED WORK

There is a large body of work related to this paper. Broadly speaking, it falls into three categories. First, others have analyzed performance of standard HPC benchmarks on public clouds. Second, there has been work in comparative usability

and cost analysis of running real scientific codes on small, medium and large scale HPC clusters against pay-as-you-go public clouds. Finally, there has been work in performance, energy and cost optimization for software components on infrastructure-as-a-service (IaaS) platforms. We focus on the first two types.

The introduction of virtualization technologies has opened up interesting possibilities for running HPC applications, such as increasing resource utilization, sandboxing and ability to customize. Youseff et al. analyzed performance of MPI applications on virtualization technologies such as Xen [31]. Ibrahim et al. studied the impact of resource customization and utilization for various virtualization techniques [18]. Cloud computing platforms incorporated these techniques to virtualize computing resources for various computation needs of individuals to enterprise scale users. Platforms such as Amazon EC2[5], FutureGrid [3] and OpenCirrus [9] make it possible to configure HPC scale resources for MPI-based distributed applications. Amazon EC2 provides a wide range of virtualized computational resources at a hourly cost per resource. FutureGrid and OpenCirrus provide medium-scale test beds for virtualized computational resources at no cost, but performance can be an issue.

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 [26], [16], [24], [13], [17], [19], [12], [11], [14] has focused on extensively benchmarking currently available EC2 cluster types with standard MPI benchmarking suites such as NAS [6] and Sequoia [2]. 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. attempt to formalize the cost of leasing CPU in HPC clusters [27]. 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. involved a detailed modeling of cost of ownership, support, and hardware and software upgrades [29]. 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. presented a comparative study of public cloud providers for different real HPC applications [21]. Again, our work differs in the use of turnaround time and cost/performance analysis at scale.

The work most closely related to our work [32] compares the cost of renting virtual machines in the cloud against

hosting a cluster. 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. Finally, we present a cost-benefit comparison between different Amazon EC2 clusters influenced by application scalability characteristics.

VII. CONCLUSION

This paper evaluated the cloud against traditional high-performance clusters along multiple dimensions. These included execution time, turnaround time, and cost. We found that from a user perspective, there are multiple considerations in choosing a cluster to run on, including raw performance, but also including the expected queue wait time along with the actual cost.

Determining a cluster on which to run is a task that we argue should be abstracted from the typical user. Our end goal is to develop tools and techniques for directing diverse sets of applications and users to the most appropriate cluster in a particular situation.

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. (LLNL-CONF-555973).

REFERENCES

- [1] ASC purple benchmarks. https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/, 2002.
- [2] ASC sequoia benchmarks. <http://asc.llnl.gov/sequoia/benchmarks/>, 2009.
- [3] Futuregrid project. <https://portal.futuregrid.org/>, 2009.
- [4] Using Moab. <https://computing.llnl.gov/tutorials/moab/>, 2012.
- [5] Amazon. Amazon web service elastic compute cloud (EC2). <http://aws.amazon.com/ec2>.
- [6] 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 *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing (SC)*, Nov. 1991.

- [7] J. Brodtkin. \$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.
- [8] J. Brodtkin. \$4,829-per-hour supercomputer built on amazon cloud to fuel cancer research. <http://arstechnica.com/business/news/2012/04/4829-per-hour-supercomputer-built-on-amazon-cloud-to-fuel-cancer-research.ars>, 2012.
- [9] R. Campbell, I. Gupta, M. Heath, S. Y. Ko, M. Kozuch, M. Kunze, T. Kwan, K. Lai, H. Y. Lee, M. Lyons, D. Milojevic, D. O'Hallaron, and Y. C. Soh. Open cirrus cloud computing testbed: federated data centers for open source systems and services research. In *Proceedings of the 2009 conference on 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 *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 169–176, 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 *Proceedings of the Second International IEEE Conference on Cloud Computing Technology and Science (CloudCom)*, pages 383–387, Nov. 2010.
- [13] C. Evangelinos and C. Hill. Cloud computing for parallel scientific HPC applications: feasibility of running coupled atmosphere ocean climate models on Amazon's EC2. In *Proceedings of the 2008 NSF/DOE Conference on Cloud Computing and its Applications (CCA)*, Oct. 2008.
- [14] 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.
- [15] Y. Gong, B. He, and J. Zhong. An overview of CMPI: network performance aware MPI in the cloud. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pages 297–298, 2012.
- [16] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn. Case study for running HPC applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 395–401, 2010.
- [17] 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 Conference on Grid Computing*, Oct. 2009.
- [18] K. Z. Ibrahim, S. Hofmeyr, and C. Iancu. Characterizing the performance of parallel applications on multi-socket virtual machines. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 1–12, 2011.
- [19] 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 *Second International IEEE Conference on Cloud Computing Technology and Science (CloudCom)*, Nov. 2010.
- [20] 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.
- [21] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: comparing public cloud providers. In *Proceedings of the 10th annual conference on Internet measurement*, pages 1–14, 2010.
- [22] D. Nurmi, J. Brevik, and R. Wolski. Qbets: Queue bounds estimation from time series. In *Workshop on Job Scheduling Strategies for Parallel Processing*, jun 2007.
- [23] 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 *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2003.
- [24] 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.
- [25] D. Singh. personal communication, Mar. 2012.
- [26] E. Walker. Benchmarking Amazon EC2 for high-performance scientific computing. *Login*, 33(5):18–23, Oct. 2008.
- [27] E. Walker. The real cost of a cpu hour. pages 35–41, 2009.
- [28] Wikipedia. Cloud computing wikipedia page. http://en.wikipedia.org/wiki/Cloud_computing, 2011.
- [29] K. Yelick, S. Coghlan, B. Draney, and R. S. Canon. The magellan report on cloud computing for science. http://science.energy.gov/~media/ascr/pdf/program-documents/docs/Magellan_Final_Report.pdf, December 2011.
- [30] S. Yi, D. Kondo, and A. Andrzejak. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, pages 236–243, 2010.
- [31] L. Youseff, R. Wolski, B. Gorda, and C. Krintz. Evaluating the performance impact of xen on MPI and process execution for HPC systems. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, 2006.
- [32] 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 *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing)*, pages 11:1–11:10, Nov. 2011.