

Scientific Computing in the Cloud

Large, virtualized pools of computational resources raise the possibility of a new, advantageous computing paradigm for scientific research. To help achieve this, new tools make the cloud platform behave virtually like a local homogeneous computer cluster, giving users access to high-performance clusters without requiring them to purchase or maintain sophisticated hardware.

Modern cloud computing platforms vary,^{1–3} but they share two critical features: they abstract the underlying compute components and they typically charge users incrementally based on their usage. The “pay-as-you-go” billing strategy isn’t new, and it has many potential advantages, especially for scientists who don’t require 24/7 accessibility. Many academic computational researchers have used shared compute facilities for decades and are accustomed to being billed per CPU-hour. What makes cloud architectures a compelling new product for scientific computing—and what differentiates them from existing supercomputing facilities—is the way they abstract the underlying compute components. These components range from hardware infrastructures to operating systems to software packages. This approach offers several advantages for scientists, users, and developers alike. First, unlike supercomputer centers, cloud hardware infrastructures give users and developers sweeping control of their clusters. This is useful when scientists have applications that require particular pieces of software to be installed at the system level. For clouds that provide software as a service, the scientist never has to install a thing. The cloud

provider installs, maintains, and optimizes the application and the scientist merely conforms to a specific API.

Here, we focus on a configuration of the Amazon elastic compute cloud (EC2; see <http://aws.amazon.com>) for scientific computation. The EC2 is part of Amazon Web services (AWS), which provides hardware infrastructure as a service. In other words, the hardware itself is abstracted into compute resources (EC2) and storage resources (simple storage service, or S3). Cloud computing services are widely used in areas such as commercial Web applications—such as Google Apps (www.google.com/apps/intl/en/business/index.html) or Microsoft’s Azure (www.microsoft.com/azure)—but have scarcely been exploited for scientific computing applications³ largely because of their differing requirements. Scientists often require platforms that are good number crunchers, for example, and virtualization software might interfere with this capability. Second, elastic scientific codes often require a high-performance network interconnect, and no cloud platform yet offers such capability.

In our study, we set out to assess EC2’s computational and network capabilities for scientific high-performance computing (HPC). In particular, we demonstrate EC2’s feasibility for ab initio electronic structure and x-ray spectroscopy modeling^{6,7} using the real-space code FEFF (<http://leonardo.phys.washington.edu/feff>), which is typical of many scientific computing applications. FEFF is a widely used scientific code that

1521-9615/10/\$26.00 © 2010 IEEE
COPUBLISHED BY THE IEEE CS AND THE AIP

JOHN J. REHR, FERNANDO D. VILA, JEFFREY P. GARDNER,
LUCAS SVEC, AND MICAH PRANGE
University of Washington, Seattle

calculates the electronic and optical properties of arbitrary, complex systems in real-space for large atom clusters, and runs on a variety of computing environments. Here, we present results that benchmark the performance of both serial and parallel versions of FEFF on EC2 virtual machines. We also describe a toolset that we developed that lets users deploy their own virtual compute-clusters, both for FEFF and other parallel codes. Finally, we explore EC2's intra- and internode communication performance using a tightly coupled scientific code and the Intel MPI Benchmarks.

Our efforts are in two main areas:

- *Development.* We created an environment that permits the FEFF user community to run different software versions in their own EC2-resident compute clusters.
- *Benchmarking.* We tested FEFF and other scientific codes performance on EC2 hardware.

To accomplish these efforts, we first had to gain an understanding of the EC2 and S3 infrastructure.

EC2 and S3 Infrastructure

The Amazon EC2 service hosts Amazon machine images (AMIs) on generic hardware located “somewhere” within the Amazon computer network. Amazon offers a set of public AMIs that users can customize to their needs, as well as several types of hardware with various performance levels. Once users select and configure an AMI, they can store it in their Amazon S3 accounts for subsequent reuse. EC2's “elasticity” denotes users' ability to spawn an arbitrary number of AMI instances while scaling the computational resources to match computational demands as needed. Currently available EC2 instances range from small (a 32-bit platform with one virtual core and 1.7 Gbytes of memory and 160 Gbytes of storage) to extra large (64-bit platform with eight virtual cores, 7 Gbytes of memory and 1,690 Gbytes of storage). These limits will likely increase in the future.

Toolsets

EC2 and S3 provide three toolsets for creating and using AMIs:

- API tools (<http://developer.amazonwebservices.com/connect/entry.jspa?externalID=368>) are command-line utilities used to bundle an image's current state and upload it to S3 storage.

- API tools (<http://developer.amazonwebservices.com/connect/entry.jspa?externalID=351>) serve as the client interface to the EC2 services, letting users register, launch, monitor, and terminate AMI instances.
- S3 libraries (<http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=47>) let developers interact with the S3 server to manage stored files, such as AMIs.

The toolsets are available in several formats, including Python, Ruby, and Java. This variety of coding languages gives developers a range of options, letting them tailor their implementations to optimize results. Currently, we use the Java implementation, supplemented with our own set of Bash scripts. For developers to use EC2, they need all three sets of tools. In contrast, users need only the API tools, unless they want to modify and store our preconfigured images. We endeavored to make these tools transparent; users need neither cloud computing nor HPC expertise to run sophisticated parallel codes on the EC2 environment.

We also experimented with Elasticfox (<http://developer.amazonwebservices.com/connect/entry.jspa?externalID=609>) and S3fox (<http://developer.amazonwebservices.com/connect/entry.jspa?externalID=366>), two GUIs that provide partial implementations of the EC2 and S3 tools. Both are extensions of the Firefox browser and provide a user-friendly alternative to the EC2 and S3 tools for AWS users. Elasticfox gives an all-in-one picture of users' current AMI states and active instances, and lets users initiate, monitor, and terminate AMI instances. S3fox mimics the interface of many commonly used FTP programs, letting users create and delete S3 storage areas. We believe these graphical browser extensions will prove useful and intuitive for the scientific-user community. Therefore, we developed our user environment to accommodate these tools. Amazon also provides its own graphical interface, the AWS management console, as a browser-independent way of managing EC2 instances.

Usability

The scientific-user community includes a growing number of investigators with parallel computation experience who are familiar with Linux and MPI, but others are virtually helpless in such HPC environments. Also, while investigators are increasingly using HPC versions of FEFF to study complex materials, many users lack access to adequate HPC resources. One of

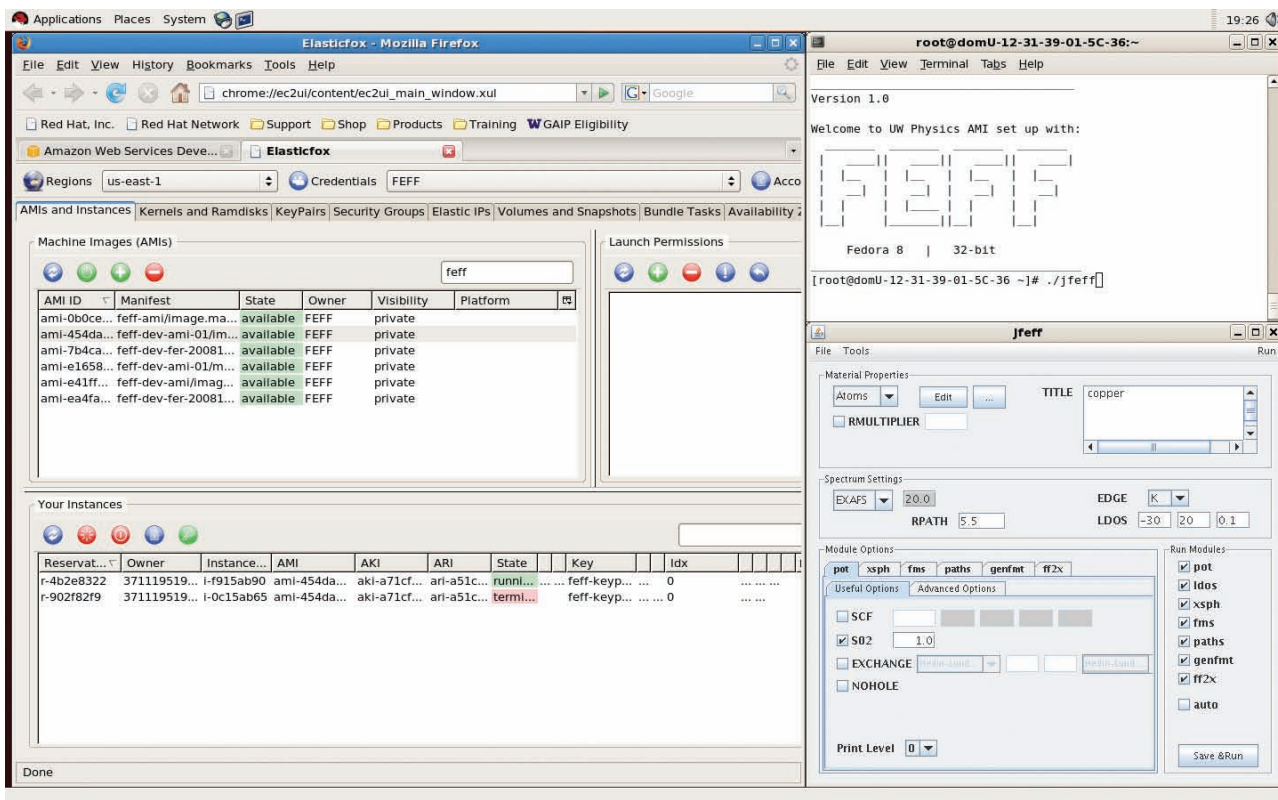


Figure 1. Amazon elastic compute cloud (EC2) for serial scientific computing. Components include the FEFF EC2 console in the upper right; the Java FEFF GUI in the lower right, and a Firefox browser running the Elasticfox extension in the background.

cloud computing’s advantages is its potential to provide such access, without requiring users to buy, maintain, or even understand HPC hardware.^{8,9}

With this in mind, we adopted a development strategy that aims to serve users lacking HPC resources. Our approach provides a complete, standalone MPI parallel runtime environment. Users need to know only their AWS account credentials; they don’t need any experience with parallel codes, hardware, or the EC2. Our environment works well for the FEFF code. However, it’s also general and can be used to launch any suitably configured Amazon EC2 Linux image for parallel computation. This gives users accustomed to using MPI on a workstation or cluster an immediately useful way to run both our FEFF MPI software and many other MPI parallel codes on the EC2.

Configuring EC2 for Serial Scientific Computing

For simplicity, we begin with an example of serial scientific computing. Starting from a public AWS AMI with a Fedora 8 Linux operating system, we created a FEFF AMI that provides both

the command-line-driven FEFF code (version 8.4) and JFEFF, a Java-based GUI that facilitates FEFF84 execution. To make these programs run smoothly, we enhanced the AMI template with X11 and a Java runtime environment. JFEFF then functions as if it were running locally. The resulting AMI occupies approximately 550 Mbytes of S3 storage. The AMI’s boot time of instances varies depending on EC2 availability but, on average, an instance boots in about two minutes and rarely takes more than three-and-a-half minutes. Figure 1 shows a screenshot of JFEFF running on EC2 and the FEFF AMI console, with the Elasticfox control screen running in the background. As might be expected, the only notable limitation we observed is the GUI’s relatively slow response when running over a network. To overcome this problem, we modified the JFEFF GUI so it can be run on a user’s local machine while the FEFF executables are resident on the EC2 instances.

Benchmarking FEFF84 Serial Performance Tests

As part of our overarching goal of achieving high-performance scientific computing in the cloud, we

first used the FEF84 AMI to test the serial performance of FEF84 on instances with different computing power.

Figure 2 shows runtime versus cluster-size results for a typical full multiple-scattering FEF84 calculation—a Boron Nitride crystal—containing between 29 and 157 atoms. This is realistic, as finite clusters with about 100 atoms are typically adequate to obtain bulk systems’ converged spectra. We used two instance types, both running 32-bit operating systems:

- a small instance using a 2.6-gigahertz AMD Opteron processor, and
- a medium instance using a 2.33-GHz Intel Xeon processor, both including FEF84 compiled with Gnu Fortran without optimization flags.

For comparison, we included results from one of the University of Washington’s local Linux systems with a 64-bit 2.0-GHz AMD Athlon processor. Figure 2 also shows the results we obtained with a highly optimized version of FEF84. We produced the compiled executable on an AMD Opteron Linux system at UW’s Department of Physics using the PGI Fortran compiler with the “-fast” optimization flag. This system is analogous to the one used in the small instance. As Figure 2 shows, the resulting code makes the small instance even faster than the medium one. Consequently, we believe that AMIs should include well-optimized HPC tools to provide good performance for scientific applications. Of course, developers can do this once and for all, so users won’t have to configure and optimize such codes for novel compute environments.

Strategies and Tools for Parallel Cloud Computing on the EC2

Setting up a virtual cluster on the EC2 is similar to setting up a physical computer cluster, but certain aspects of the EC2 infrastructure pose unique challenges. In most physical clusters, for example, the administrator typically has complete control of the node IP addresses, but on the EC2, they’re dynamically allocated at boot time. So, you must gather this and other information before you can configure a virtual cluster. Security is also quite different. For example, to reduce a cluster’s vulnerability, access certificates aren’t stored within the AMI and are only transferred during setup.

These challenges, together with our desire to make the compute-cluster setup transparent to users, steered us toward a cloud-cluster structure that’s slightly different from most typical physical

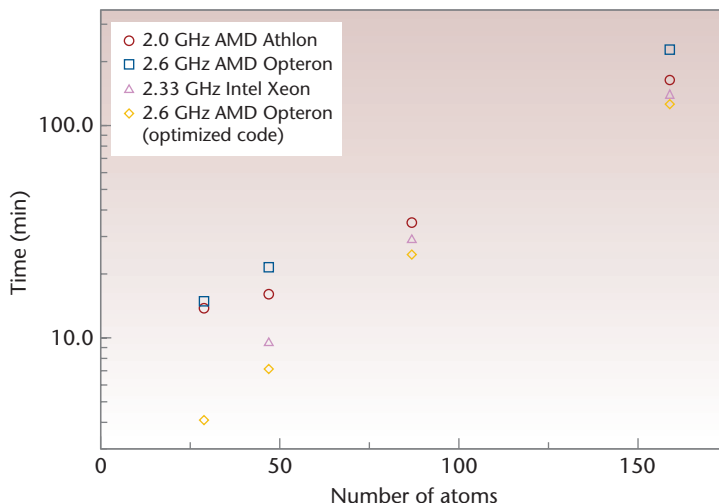


Figure 2. Comparing serial performance (runtime versus cluster size) of different AMI instance types for typical FEF84 x-ray spectra calculations. Overall EC2 virtual performance is similar to that of a physical system, with the medium instance being about twice as fast as the small one. Optimized code in the small instance performs better than non-optimized code on the medium one.

compute-clusters in three key ways. First, because Amazon charges for all instances booted regardless of CPU load, we did away with the usual server head node of physical clusters. Thus cloud clusters are composed of only compute nodes. However, we do designate one of the nodes as a common disk server because many parallel codes (such as FEF84) require shared disk access for all parallel processes. Second, we eliminated the multiuser cluster concept. The cloud clusters have a single user, specifically set up to run the payload program. Finally, users can launch as many clusters as they need to run several simulations simultaneously. Our EC2 cloud-cluster implementation is homogeneous, thus facilitating parallel task load balancing. Figure 3 shows the resulting cloud-cluster scenario.

EC2 Compute-Cluster Tools

As we noted earlier, AWS provides Java and Python tools designed to interact with EC2. Based on the Python modules, other developers produced a set of scripts for managing MPI clusters on EC2 (www.datawrangling.com/mpi-cluster-with-python-and-amazon-ec2-part-2-of-3). However, these scripts have many limitations, and changes in Amazon’s API reporting format have rendered some unusable.

Given the scripts’ initial success, we developed a new set of tools¹⁰ aimed at typical MPI users interested in HPC calculations on complex systems. We tested the tools on the FEF84 AMI.

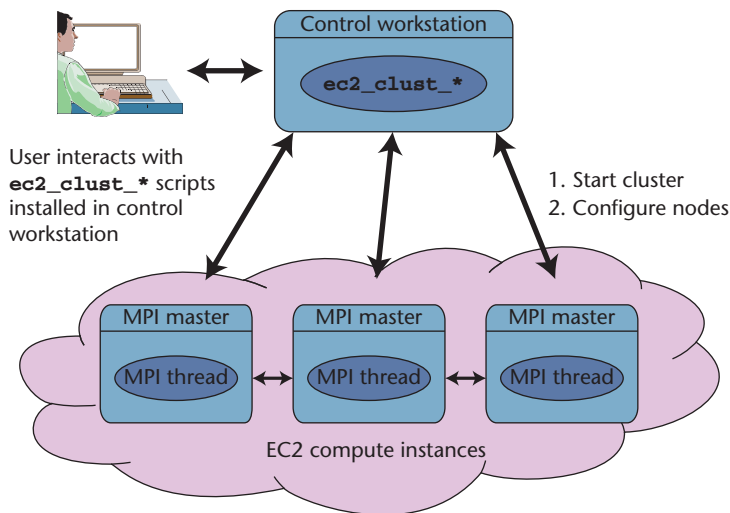


Figure 3. A typical cloud cluster generated by the EC2 cluster tools. Lacking a head node and using a single user-configured script to run the payload program, this cluster differs from typical clusters. Many copies of these cloud clusters can be used to run simultaneous simulations.

They consist of a set of Bash scripts that perform the basic tasks involved in interacting efficiently with the cloud cluster. These tasks include launching, connecting to, transferring files to and from, monitoring, and finally terminating a cloud cluster. The scripts are easy to install and have few requirements:

- the Java EC2 API,
- the Java runtime environment (RTE), and
- a *NIX environment with the Bash and secure shells (ssh).

Although not essential, we also encourage users to install the Elasticfox extension of the Mozilla Firefox browser. This extension provides a user-friendly monitor of the user's instances on the cloud, which might help avoid unnecessary charges for runaway instances. Currently, the EC2 cloud-cluster tools can be installed either per user or on server mode. In the near future, we plan to offer a specially configured EC2 AMI containing all the software required to launch a cloud cluster. This will eliminate tool and security certificate installation, thus further simplifying cloud-cluster interactions.

The toolset's main starting script is `ec2_clust_launch`, which sets up an EC2-cluster with N nodes. Figure 4 shows a typical launch sequence log for a cluster with two instances. First, this script requests N instances, parsing and storing the EC2 reservation's information. Then, `ec2_clust_launch` monitors the reservation's

status until all instances have booted. Once the full reservation is running, we gather the required internal EC2 IP addresses, create the configuration files required to run MPI applications, and distribute them to all nodes. On most physical clusters, all the nodes share storage, which is usually mounted from a designated file server node. Here, we assign the instance with boot index 0 as this server and export and mount a scratch area on all nodes. We accomplish this using the standard network file system (NFS). In the final step, we transfer the secure shell (ssh) key files, used to connect to the cluster for computing purposes. We store all information about the cluster in a per-user database, which contains information that the other cluster tools will use. Several clusters can be launched simultaneously, each having a unique identifier given by the EC2 reservation ID or a user-assigned label. All scripts can be directed at a specific cluster by using the user-assigned label or the reservation ID. If neither is given, the scripts default to the last cloud cluster created. Users can access a list of active cloud clusters through the `ec2_clust_list` command.

Once the cluster is fully booted and ready, users can connect to the master node—that is, the node with the boot index 0—using either the `ec2_clust_connect` or the `ec2_clust_connect_root` scripts. These scripts are wrappers of the `ssh` command; the former is intended for computing purposes and the latter for administration purposes only. These administration tasks might be required for simple session adjustments. Because all configuration changes are lost upon cluster termination, users should perform any complicated image configuration changes on the master image and then save it using the AMI tools. From within the master node, users can access slave nodes with a password-less `ssh` command. Files and directories can be recursively transferred to and from the cluster using the `scp` wrapper scripts `ec2_clust_put` and `ec2_clust_get`, respectively. We've transparently included all security certificates and IP addresses that the connection and file transfer scripts require, thus hiding all security exchanges. Finally, the `ec2_clust_terminate` script shuts down all instances associated with a cluster.

We illustrate the process using the parallel code FEFF84 MPI. Once users log in to a given cluster, the FEFF MPI runtime environment is ready to use. They can access NFS shared storage through `/mnt/share`, and a simple script that specifies the executable code (`rfeff84g77mpi`) launches all the daemons that the MPI

```

(latte) ~/Cloud/Cluster/Test [16] ec2_clust_launch 2
Preparing to launch 2 instances
Launched cluster with reservation ID: r-dffd55b6
Instance IDs: i-efd65286 i-eed65287
Booting...
. 0 .. 0 .. 0 .. 0 .. 0 .. 0 .. 1 .. 1 .. 2 .
All instances ready
Head node IP: ec2-174-129-150-120.compute-1.amazonaws.com
Creating a list of internal IP addresses
Creating host files
Copying host files to all instances
.ec2_clust_hosts          100% 196      0.2KB/s  00:00
.ec2_clust_hostfile       100% 18       0.0KB/s  00:00
.ec2_clust_hosts          100% 196      0.2KB/s  00:00
.ec2_clust_hostfile       100% 18       0.0KB/s  00:00
Creating shared directories
Creating exports file
Copying exports file to head instance
.ec2_clust_exports        100% 192      0.2KB/s  00:00
Restarting nfs on head instance
Adding mount information to fstab on slave instances
Mounting shared directories on slave instances
Transferring user information to all instances
feff_use_pub.pem          100% 609      0.6KB/s  00:00
feff_use_prv.pem          100% 668      0.7KB/s  00:00
feff_use_pub.pem          100% 609      0.6KB/s  00:00
feff_use_prv.pem          100% 668      0.7KB/s  00:00
Saving information in local cluster configuration file
Setup is finished
Your cluster is ready

```

Figure 4. Output example showing the boot and configuration sequence of a representative cloud cluster with two instances run from a local machine (latte). The script launches the required instances and waits until they have booted. Then it gathers the information needed to setup file-sharing and configures MPI. Finally, it transfers the security certificates used to access the cluster.

implementation requires. A cluster includes two other scripts: `ec2_clust_usage` is intended to monitor processes using more than 5 percent of the CPU on each node; `ec2_clust_load` reports the CPU load averages.

Although we originally designed these scripts for the parallel FEFF AMI, many MPI programs share a runtime infrastructure similar to that of FEFF; hence, users can easily modify the tools for use with other parallel codes. These tools let us turn a conventional laptop into a machine and console that controls a virtual supercomputer.

Benchmarking FEFF84 Parallel Performance

Most HPC platforms offer a high-performance network interconnect to increase the parallel codes' ability to scale to numerous nodes. In contrast, EC2's nodes have only a basic ethernet connection. Consequently, we began our study of parallel scalability with a code that doesn't place great demands on the network in terms of either latency or bandwidth. The parallelized FEFF84 code ("FEFF84 MPI") is largely

a data-parallel computation, where the messages passed between nodes are for coarse-grained synchronization of tasks. The sizes of the messages are quite small, meaning that any departure from linearity is due to issues of latency and load balancing. This gives EC2 the best chance of successfully achieving scalability on a computational physics code, and it also lets us investigate the infrastructure required to automatically launch and administer parallel MPI calculations in the cloud.

Figure 5 shows scaling versus the number of CPUs observed in EC2 compared to a local 1.8-GHz AMD Opteron HPC cluster with gigabit ethernet in UW's Department of Physics. The comparable behavior shows that using a virtual EC2 cluster doesn't significantly degrade FEFF84's parallel performance compared to that of a conventional physical computer cluster with standard networking. In fact, the cloud-cluster performance is slightly better due to the larger memory per processor available in the EC2 instances. This is especially noticeable for the values with 48 processors.

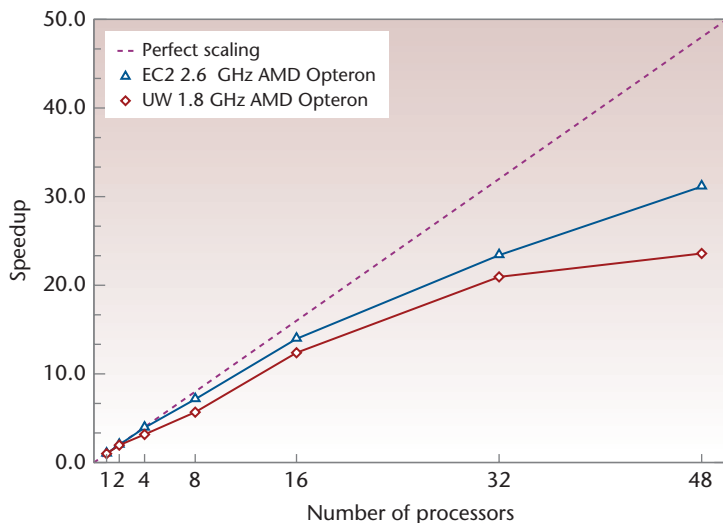


Figure 5. The scalability of FFFF84MPI for the EC2 compared to a local 1.8 GHz Opteron cluster at the University of Washington. The scaling for the virtual and physical clusters is very similar, showing that the virtualization doesn't degrade FFFF84MPI performance. Overall, the EC2 cluster provides better performance largely due to better memory availability.

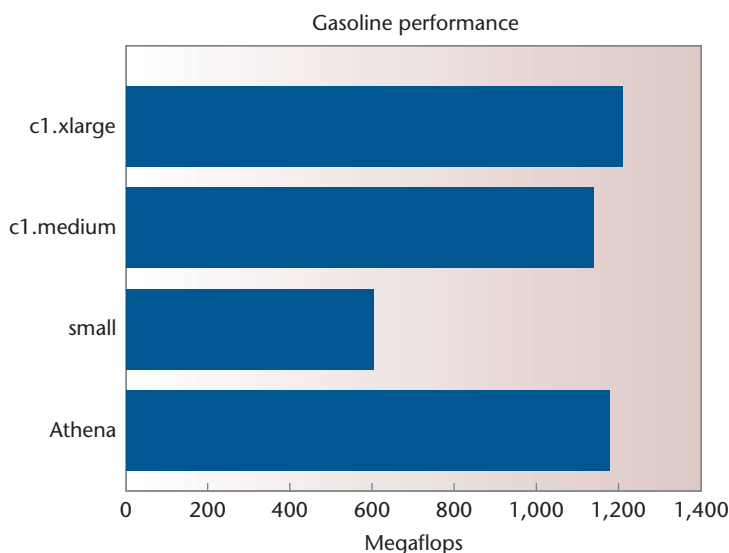


Figure 6. Gasoline's serial performance on the Amazon high-CPU extra-large (c1.large), medium (c1.medium), and small instances. Athena is a local HPC Linux cluster at the University of Washington.

Additional Benchmarks: Tightly Coupled Calculations

To better quantify EC2's performance characteristics, we also used an existing benchmarking code from astrophysics. This let us compare EC2 to other high-end computing (HEC) systems. The benchmark is based on Gasoline,¹¹ an n -body gravitational code used to simulate large-scale structure, galaxy evolution, and planet formation.

Because the force of gravity is long range, calculating the gravitational force exerted on a single object in the simulation requires knowledge of the entire simulation dataset. Also, because this dataset is typically spread across thousands of computational nodes, considerable message passing is required for this kind of calculation. Given this, gravity codes like Gasoline perform best on systems with high-performance networks. The message patterns are largely random and asynchronous, with messages several kilobytes in size. Therefore, Gasoline's parallel performance is greatly affected by network latency, but relatively insensitive to bandwidth. Its serial performance is also driven largely by memory latency (not bandwidth, like most scientific codes), which could be affected by a virtual machine hosting the application's increasing layers of indirection.

We conducted tests on several sets of EC2 compute instances as well as on current HEC platforms. Our main test system was the Department of Physics' local Athena Linux cluster, a 10-teraflops (Tflops) cluster consisting of 140 compute nodes connected via a 4-gigabits-per-second InfiniBand Double Data Rate (DDR) fabric. Each compute node has two quad core Intel 2.33-GHz E5345 Xeon processors and 8 Gbytes of RAM. We also compared benchmarks to the Cray XT3 at the Pittsburgh Supercomputing Center, a classic HEC-capability platform designed to provide maximum performance and scalability to thousands of processors. At the time we performed our benchmarks, the Cray had 2,048 2.4-GHz AMD Opteron processors connected by a Cray SeaStar fabric.

Serial performance. Figure 6 shows Gasoline's serial performance on the local Athena cluster and three different EC2 instance types. All tests used the same version of the Gnu C compiler. By examining `/proc/cpuinfo` on each instance at the time of the benchmark, we could discover the underlying hardware for that run. Amazon doesn't guarantee physical specifications for instance hosts, so the hardware on which our instances were running probably represents a subset of what could have been allocated. EC2 assigned

- the small instance, half a core of a 2.6-GHz dual-core AMD Opteron 2218HE processor;
- the medium high-CPU instance (c1.medium), two of the four cores of a 2.33-GHz quad-core Intel Xeon E5345 system; and
- the extra-large high-CPU instance (c1.large), an entire node with two quad-core E5345s.

The Athena cluster had the exact same CPUs as the EC2 instance hosts. As Figure 6 shows, Gasoline’s serial performance on the c1.medium and c1.xlarge instances was comparable to the “bare-metal” runs on Athena. In fact, the c1.xlarge instance actually performed slightly better than an Athena node. The Xeon E5345 has been deployed with a variety of (DDR2) memory speeds, and it’s possible that the c1.xlarge instance might have had faster memory than the Athena nodes. At the time we measured it, Amazon described the small instance as having performance comparable to “a 1.0–1.2 GHz 2007 Opteron or Xeon” class processor. Because neither the Opteron nor the Xeon was produced with clock rates as slow as 1.0 or 1.2 in 2007, we surmise that Amazon really means performance comparable to half of a 2.0- to 2.4-GHz processor. This seems to be the case, as the small instance performs at a level of about what we’d expect from 50 percent of a 2.4-GHz core, even though it’s hosted by a 2.6-GHz CPU. Consequently, it seems that some overhead is introduced by the virtualization process when processor cores are shared, less overhead is introduced if only nodes are shared (but cores aren’t), and almost no overhead is present given an entire physical node. From a serial perspective, EC2 instances appear to offer performance comparable to bare metal for scientific HEC applications.

Parallel performance. We also conducted strong scaling tests—both within a single compute node to test memory contention, and between multiple nodes as a test of network performance. In all cases, we assigned each core exactly one MPI process. Figure 7 shows results of both tests. The solid blue and red lines show the scaling up to eight cores within a single Athena node and the c1.xlarge instance, respectively. Performance is comparable, indicating that the virtual machine hosting environment introduced little overhead, even when all eight cores were running at full capacity. Gasoline achieves similar scalability on the Cray XT3 (blue line), although this system is using a high-performance network interconnect. The dashed lines indicate runs performed across multiple Athena and EC2 nodes. In this case, we see a definite advantage of an HEC system over EC2. Scalability is already beginning to suffer at eight EC2 nodes.

Network Characteristics

Our further investigations with Gasoline and the Intel MPI Benchmarks (<http://software.intel.com/>

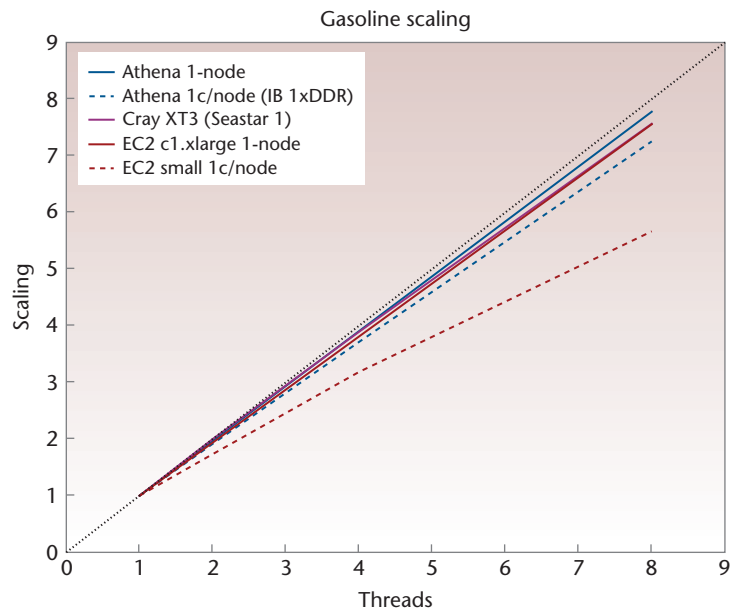


Figure 7. Strong scaling on EC2 and high-end computing platforms. The similar behavior of the Athena system and the c1.xlarge instance indicates that virtualization introduces little overhead. Performance degradation appears when the simulations are run over the network (dashed lines).

en-us/articles/intel-mpi-benchmarks) showed latency and bandwidth characteristics consistent with the network performance achievable when running across workstations in a building, connected by typical 100-Mbps or 1-Gbps networks.

For example, we used the PingPong benchmark to measure both bandwidth and latency between two nodes. The UW HPC Athena cluster, which has a high-performance InfiniBand DDR interconnect, achieves a latency of 3 to 4 microseconds and a bandwidth in excess of 1 Gbps on this benchmark. A typical modern academic building wired with a 100-Mbps or gigabit network will see latencies between approximately 50 and 150 μ s and a bandwidth between 10 and 100 Mbytes per second. We tried four different pairs of small instances in the EC2 availability zone “us-east-1a” and found best-case latencies—that is, the smallest latency reported by any packet size during any PingPong run between two instance pairs—ranging from 160 to 220 μ s. This is about what we’d expect for instances in a large building separated by several network hops. Best-case bandwidth ranged from 35 to 45 Mbytes per second, indicating the likely presence of a gigabit network.

All things considered, it’s comforting to note that EC2’s network performance wasn’t worse. Although we ensured that all instances were in the same AWS “availability zone,” it’s possible that they could’ve been spread across different buildings or even separate sites. So, as far as

on-node performance is concerned, EC2 is apparently a viable platform for hosting many scientific applications. It can also host parallel applications, provided their performance isn't degraded by network performance similar to that in a generic building-wide research computing scenario.

Remaining Challenges


Our results show that scientific cloud computing is generally feasible and has advantages for many users, such as those with moderate computational requirements that aren't satisfied with a simple workstation, but lack access to large supercomputer facilities. Nevertheless, several issues remain. For instance, it can be difficult to assess a priori whether it's more efficient to use a cloud cluster than to acquire and maintain the hardware to fulfill those requirements.¹² The choice depends on several parameters—including, among others, the accounting system's granularity for various EC2 instances, the target software's parallel coupling level and single processor performance, and the typical turnaround time at supercomputer centers.

With these challenges in mind, several aspects of scientific cloud computing remain to be addressed. It would be especially interesting to develop even higher performance computing AMIs that would use 64-bit operating systems and include high-performance compilers and numerical libraries. Fortunately, cloud computing is developing rapidly and steady improvements along these lines are being made, as in the AWS High-CPU extra-large instance. These images could provide a representative production environment to assess the overall economic feasibility of high-performance scientific cloud computing. The automated cluster tools should also facilitate further tests aimed at optimizing interprocess communications. As we described earlier, this is a minor issue for a moderately coupled program like FEFF, but is likely to be important for many other high-performance scientific codes.

One of the main lessons learned from our attempts at scientific cloud computing is that, heretofore, the main roadblock to using cloud computing effectively for many scientific users is the rather alien environment presented by the cloud compared to typical local clusters. The need to overcome this roadblock was our primary motivation for developing a set of cluster tools that hide as much of the cloud environment as possible and turn it transparently into a system that is more familiar to many computational

physicists. In the future, we can extend these simplifications by modifying the tools to run from a special control AMI. Within this setup, users wouldn't have to download the scripts to their workstations. Instead, they'd simply request a control instance, and then launch and manage their MPI clusters from it. This extension would require the development of additional easy-to-use tools to transfer files both to and from the control instance, as well as special care regarding access security. Finally, we envision a unified front end to manage MPI clusters on the EC2, similar to Elasticfox. Such a front end would have the double advantage of removing the need for command-line utilities and making the tools more platform-independent.

The AWS EC2 is reasonably adaptive to our goal of making high-performance scientific computation readily available to the scientific community. We've explored a variety of promising methods that let users interact with custom AMIs—ranging from predefined scripts that manage AMIs locally to tools that let users remotely `ssh` into the instances and control them directly. This virtualization lets code developers optimize and preinstall scientific codes on AMIs, thus facilitating control over the computational environment.

In terms of performance, we've demonstrated that the EC2 cloud clusters can provide access to reliable, high-performance computation for general scientific users without requiring that they purchase and maintain hardware on their own, provided that their application doesn't demand high-performance network interconnects. Serial performance of scientific codes was comparable to bare-metal runs on similar hardware. However, the network that connects the EC2 compute hardware in the same Amazon availability zone has similar latency and bandwidth characteristics to a gigabit Ethernet network in a large office building. Although network performance could've been even worse (Amazon offers no guarantees), it's still a far cry from the capability of the high-performance interconnects found on most academic high-end computing clusters. 

Acknowledgments

The US National Science Foundation grant DMR-0848950 and NSF CDI grant PHY-0835543 (JG and FV) support this work, and the US Department of Energy grant DE-FG03-97ER45623 supports the FEFF

project. We thank Chuck Bouldin, Richard Coffey, Ed Lazowska, and Chance Reschke for comments, and especially Deepak Singh and Werner Vogels for their support of this project and for providing initial AWS resources. Some of the tools developed in this project were inspired by the MPI Cluster Tools developed by Peter Skomoroch (Datawrangling); the JFEFF GUI was adapted from that of David Bitseff. Cray XT3 benchmarks were produced using allocations of advanced NSF-supported computing resources operated by the Pittsburgh Supercomputing Center.

References

1. L.M. Vaquero et al., "A Break in the Clouds: Towards a Cloud Definition," *ACM Sigcomm Computer Comm. Rev.*, vol. 39, no. 1, 2009, pp. 50–55.
2. F. Sullivan, "Cloud Computing for the Sciences," special issue, *Computing in Science & Eng.*, vol. 11, no. 4, 2009, pp. 10–11.
3. M. Armbrust et al., *Above the Clouds: A Berkeley View of Cloud Computing*, tech. report UCB/EECS-2009-28, Electrical Eng. and Computer Sciences Dept., Univ. of California, Berkeley, 10 Feb. 2009; www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.
4. C. Hoffa et al., "On the Use of Cloud Computing for Scientific Workflows," *Proc. 4th IEEE Int'l Conf. on eScience*, IEEE CS Press, 2008, pp. 640–645.
5. C. Evangelinos and C.N. Hill, "Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere–Ocean Climate Models on Amazon's EC2," *Proc. Cloud Computing and Its Applications*, 2008; <http://cca08.org/papers.php>.
6. A.L. Ankudinov et al., "Real Space Multiple Scattering Calculation and Interpretation of X-Ray Absorption Near Edge Structure," *Physical Review B*, vol. 58, no. 12, 1998, pp. 7565–7576.
7. A.L. Ankudinov et al., "Parallel Calculation of Electron Multiple Scattering Using Lanczos Algorithms," *Physical Review B*, vol. 65, no. 10, 2002, pp. 104107–104118.
8. J. Napper and P. Bientinesi, "Can Cloud Computing Reach the TOP 500?" *Proc. Workshop Unconventional High-Performance Computing*, ACM Press, 2009, pp. 17–20.
9. E. Walker, "Benchmarking Amazon EC2 for High-Performance Scientific Computing," *login.*, vol. 33, no. 5, 2008, pp. 18–23.
10. F. Vila, J.J. Rehr, and J. Gardner, "Bash Scripts for Scientific CC," tech. report, Dept. of Physics, Univ. Washington, forthcoming, 2010.
11. J.G. Stadel, *Cosmological N-body Simulations and Their Analysis*, doctoral thesis, Dept. of Astronomy, Univ. Washington, 2001.
12. D. Kondo et al., "Cost-Benefit Analysis of Cloud Computing Versus Desktop Grids," *Proc. Int'l Parallel and Distributed Processing Symp.*, IEEE CS Press, 2009, pp. 1–12.


John J. Rehr is a professor of physics at the University of Washington, Seattle. His research interests include condensed-matter theory, theoretical spectroscopy, excited states and response functions, and scientific computing. Rehr has a PhD in theoretical physics from Cornell University. He's a member of the American Physical Society and the International XAFS Society. Contact him at jjr@uw.edu.

Fernando D. Vila is a research associate professor of physics at the University of Washington, Seattle. His research interests include quantum chemistry, theoretical spectroscopy, electronic structure of molecular systems, intermolecular interactions, and scientific computing. Vila has a PhD in chemistry from the University of Pittsburgh. He is a member of the American Physical Society. Contact him at fdv@uw.edu.

Jeffrey P. Gardner is a senior research scientist for high-performance computing in the Department of Physics, University of Washington. His research interests include parallel computing, data-intensive scalable computing, cosmology, large-scale structure, scientific cloud computing, scalable data analysis, and parallel programming paradigms. Gardner received a PhD in astronomy from the University of Washington. Contact him at gardnerj@phys.washington.edu.

Lucas Svec is a physics graduate student at the University of Washington. His research interests include quantum computation and scientific computing. Svec has a BS in physics and math from the University of Washington. Contact him at svecl@u.washington.edu.

Micah Prange is currently a research associate in the Department of Physics and Astronomy at Vanderbilt University, Nashville. His research interests include theoretical electron scattering theory and scientific computing spectroscopy. Prange received a PhD in physics from the University of Washington. He is a member of the American Physical Society. Contact him at micah.prange@gmail.com.

 Selected articles and columns from IEEE Computer Society publications are also available for free at <http://ComputingNow.computer.org>.