Scientific Programming 19 (2011) 107–119
DOI 10.3233/SPR-2011-0324
IOS Press

Performance and cost analysis of the Supernova factory on the Amazon AWS cloud

Keith R. Jackson [*], Krishna Muriki, Lavanya Ramakrishnan, Karl J. Runge and Rollin C. Thomas
Lawrence Berkeley National Lab, Berkeley, CA, USA

Abstract. Today, our picture of the Universe radically differs from that of just over a decade ago. We now know that the Universe is not only expanding as Hubble discovered in 1929, but that the rate of expansion is accelerating, propelled by mysterious new physics dubbed "Dark Energy". This revolutionary discovery was made by comparing the brightness of nearby Type Ia supernovae (which exploded in the past billion years) to that of much more distant ones (from up to seven billion years ago). The reliability of this comparison hinges upon a very detailed understanding of the physics of the nearby events. To further this understanding, the Nearby Supernova Factory (SNfactory) relies upon a complex pipeline of serial processes that execute various image processing algorithms in parallel on ∼10 TBs of data.

This pipeline traditionally runs on a local cluster. Cloud computing [Above the clouds: a Berkeley view of cloud computing, Technical Report UCB/EECS-2009-28, University of California, 2009] offers many features that make it an attractive alternative. The ability to completely control the software environment in a cloud is appealing when dealing with a community developed science pipeline with many unique library and platform requirements. In this context we study the feasibility of porting the SNfactory pipeline to the Amazon Web Services environment. Specifically we: describe the tool set we developed to manage a virtual cluster on Amazon EC2, explore the various design options available for application data placement, and offer detailed performance results and lessons learned from each of the above design options.

Keywords: Cloud computing, AWS cloud, Supernova factory

1. Introduction

The goal of the Nearby Supernova Factory (SNfactory) experiment is to measure the expansion history of the Universe to explore the nature of Dark Energy with Type Ia supernovae, and also to improve our understanding of the physics of these events to improve their utility as cosmological distance indicators. Operating the largest data-volume supernova survey from 2004 to 2008, SNfactory made and continues to make heavy use of high performance computing. SNfactory maintains and operates a complex software pipeline using a local PBS cluster. However, as the volume of data increases, more resources are required to manage their application.

Cloud computing is an attractive alternative for this community of users for a variety of reasons. The pipeline consists of code packages developed by members of the community that have unique library and platform dependencies (e.g., preference of 32 over 64 bit to support legacy code). This often makes it difficult to run in shared resource environments like supercomputing centers where the software stack is pre-determined. In addition, the signal extraction algorithms that are a major component of the pipeline are constantly evolving and users need the ability to use a fixed environment and make minor changes before running a new experiment. Thus, providing access to a shared environment to collaborators is critical to the user community.

Cloud computing provides the ability to control software environments, allows users to control access to collaborators that need to access a particular setup and enables users to share environments through virtual machine images. These features address some of the challenges faced by the science users today. The requirements for the SNfactory community are not unique and represent the needs of a number of scien-

[*]Corresponding author: Keith R. Jackson, Lawrence Berkeley National Lab, 1 Cyclotron Road MS:50B-2239, Berkeley, CA, USA. Tel.: +1 510 486 4401; Fax: +1 510 486 6363; E-mail: krjackson@lbl.gov.

tific user communities. Earlier studies have evaluated performance implications for applications on cloud environments and experimented with specific choices of deployment [10,11,13,16]. However, significant effort is required today for scientific users to use these environments and it is largely unclear how applications can benefit from the plethora of choices available in terms of instance types and storage options.

In the context of the SNfactory we study the feasibility of Amazon Web Services (AWS) [2] as a platform for a class of scientific applications, and detail the impact of various design choices. Specifically, (a) we describe the tool set we developed to manage a virtual cluster on Amazon EC2, (b) we explore the various design options available for application data storage, (c) we offer detailed performance results and lessons learned from each of the above design options.

Our paper is structured as follows: Section 2 provides background on the SNfactory application. Section 3 describes the design decisions made to port the SNfactory data pipeline to AWS. Section 4 details the experiments run, and their results. Section 5 describes a series of microbenchmarks run to better understand the application level performance. Section 6 describes related work, and we present our conclusions in Section 7.

2. Background

Cosmology is the science of mapping out the history of the Universe back to the instant of its creation in the Big Bang. A complete history of the Universe spells out the origin of matter, formation of galaxies and clusters of galaxies, and the dynamics of space–time itself. Cosmology sets the stage for the story that explains our species' place in the Universe.

Today, our picture of the Universe radically differs from that of just over a decade ago. We now know that the Universe is not only expanding as Hubble discovered in 1929, but that the rate of expansion is accelerating. This revolutionary discovery was made by comparing the brightness of nearby Type Ia supernovae (which exploded in the past billion years) to that of much more distant ones (from up to seven billion years ago). The more distant supernovae appeared dimmer than expected, and repeated experiments since the initial discovery have confirmed that the excess dimming is due to acceleration, propelled by new physics dubbed “Dark Energy” [17,18].

Type Ia supernovae are exploding stars – as bright as an entire galaxy of normal stars – whose relative brightness can be determined to 6% accuracy. They arise from a white dwarf star that accretes gas from a companion star and explodes around a critical mass equal to 1.4 times that of our Sun. This standard amount of fuel makes for a “standard bomb”, hence Type Ia supernovae make excellent distance indicators due to the relatively small dispersion in brightness. Understanding the origin of these supernovae, how they explode, and how to better calibrate them as distance indicators is the goal of the US/France Nearby Supernova Factory experiment [1].

The SNfactory operated the largest data-volume supernova survey active during 2004–2008, using the QUEST-II camera on the Palomar Oschin 1.2-m telescope managed by the Palomar-QUEST Consortium. Typically, over 50 GB of compressed image data was obtained each night. This data would be transferred from the mountain via radio internet link (on the High-Performance Wireless Research and Education Network) to the San Diego Supercomputing Center, and from there to the High Performance Storage System (HPSS tape archive) at the National Energy Research Scientific Computing Center (NERSC) in Oakland, California. The next morning, the night's data were moved to the Parallel Distributed Systems Facility (PDSF) cluster for processing, reduction and image subtraction. Software identified candidates in subtractions using a variety of quality cuts, and later, machine learning algorithms to identify real astrophysical transients and reject image/subtraction artifacts. Humans performed the final quality assessment step, saving and vetting candidates using historical and context data using a custom scientific workflow tool, SNwarehouse [7]. The entire process, from the start of data collection to human identification of candidates, took approximately 18 h.

Candidates in SNwarehouse were scheduled for follow-up on the SNfactory's custom-designed, and custom-built, SuperNova Integral Field Spectrograph (SNIFS) installed permanently on the University of Hawaii (UH) 2.2-m telescope atop Mauna Kea. The SNIFS instrument and UH 2.2-m telescope were remotely controlled over a Virtual Network Computing (VNC) interface, typically from France where daytime corresponds to Hawaiian nighttime. An agreement with the University of Hawaii ensured that the SNfactory had 30% of the total observing time on the UH 2.2-m for its supernova follow-up program. The result is that the SNfactory collaboration discovered, observed in detail, and is currently analyzing a new set of Type Ia supernovae in greater numbers and, through SNIFS, with greater precision than was ever possible before. Over 3000 spectroscopic observations of nearly 200 individual supernovae are now ready for study, and it is possible that SNIFS could be operated beyond 2012, meaning that new data processing will be needed over the next several years. This data set represents a wholly new potential for understanding Type Ia supernovae and using them to better measure the properties of Dark Energy.

SNIFS was designed for the express purpose of obtaining precise and accurate spectrophotometric time-series observations of supernovae. This strategy is in marked contrast to other supernova surveys which utilize spectroscopy primarily as a one-shot means of classifying objects, and rely on multi-band photometric imaging observations (a lower-dimensionality data set) as principal science product. The challenge of precise spectrophotometry dictated SNIFS design and the follow-up strategy. For example, SNIFS basic design is that of an integral field spectrograph, meaning that the entire field of view containing a supernova is segmented into spatial chunks that are each dispersed to produce a spectro-spatial data-cube. Most supernova spectroscopy is performed with a slit spectrograph, where care must be taken to avoid wavelength dependent slit-loss of photons that the SNIFS design avoids completely by capturing all the supernova light. Also, the decision to pursue guaranteed telescope time and permanently mount SNIFS means that SNfactory has unfettered access to it at almost any time for any type of calibration experiment that can be conceived and executed remotely. However, the custom design of the instrument means that the extensive software code-base for reduction and analysis of the data is designed and implemented by collaboration members (scientists, postdocs, and students), from robotic instrument/telescope control to data reduction and analysis. Several candidate implementations for various steps in data reduction may need extensive validation before one is selected (alternatives developed may also be held in reserve for hot-swapping). The general picture of the SNIFS data reduction "pipeline" (referred to here as "IFU") seems quite typical of many other scientific collaborations which depend heavily on software, computing, and data analysis with custom instrumentation: Codes and scripts, running in a Linux/Unix batch-queue environment, controlled by scripting (e.g., Python) wrappers that coordinate work through a database or meta-data system.

SNIFS reduction tasks include standard CCD image preprocessing (bias frame subtraction to remove electronics artifacts, flat-fielding to map pixel-to-pixel response variations and bad pixels, scattered-light background modeling and subtraction), wavelength and instrument flexure corrections (solving for 2D instrument distortions using arc-lamp exposures), mapping 2D CCD pixel coordinates into 3D (wavelength, x, y) data cubes. Low-level operations include digital filtering, Fourier transforms, full matrix inversions, and nonlinear function optimization. These lower level operations are mostly performed in a mixture of C, C++, Fortran and Python. The current raw data set is approximately 10 TB, but after processing this balloons to over 20 TB and is expected to continue to grow. The pipeline is heavily dependent on external packages such as CFITSIO [9], the GNU Scientific Library (GSL) [12], and Python libraries like scipy and numpy (which in turn also depend on BLAS citeblas, LAPACK [6], etc.). The whole pipeline is "process-level" parallel. Individual codes are not parallel, so parallelism is achieved by running large numbers of serial jobs to perform the same task using different inputs. Since late 2007 the SNIFS flux-calibration pipeline has been running on a large Linux cluster at the IN2P3 Computing Center in Lyon, France – a facility shared with a number of other high-energy physics experiments, most notably ones at the Large Hadron Collider (LHC).

The SNfactory's dependence on large, shared Linux clusters at NERSC and CCIN2P3 revealed a number of previously unanticipated issues. In both cases, $24/7$ support (especially weekends) is unavailable except in cases of emergency – and what constitutes an emergency to a scientific experiment may not register as such to cluster management personnel. This issue could be ameliorated if each experiment simply managed its own mid-sized or large cluster, but this would obviate the economy of scale gained through a central computing resource. A compromise would be to give users root or privileged access to the system, but security problems obviously rule that out. Also, decisions made by cluster management are necessarily driven by general policy and cannot be easily tailored to fit every need of every experiment. For example, at CCIN2P3, the entire operating system and software architecture is rolled over roughly every 18 months – this change is not transparent to users, and experiments without a cadre of software experts must draft their scientists into debugging and rewriting lines of code just to adjust to newly added or changed dependencies. A cynical interpretation by scientists of this practice might be "if it ain't broke, break it", but these users generally recognize the benefit of adapting to new systems and architectures, but want to make those changes when they can afford to and can profit. From a financial standpoint, using scientists to debug code is a suboptimal allocation of limited funds.

Because of these issues and others, the SNfactory seized the opportunity to experiment with cloud computing and virtualization in general, with the Amazon Elastic Compute Cloud (EC2) [4]. Key aspects that were attractive to SNfactory included:

- Ability to select any flavor of Linux operating system.
- Options for architecture: 32-bit operating systems for legacy code.
- Capability to install familiar versions of Linux binary packages.
- Capacity to conveniently install third-party packages from source.
- Access as super-user and shared access to a "group" account.
- Immunity to externally enforced OS or architecture changes.
- Immediate storage resource acquisition through EBS and S3.
- Economy of scale and reliability driven by market demands.

3. Design

Porting a scientific application like the SNfactory pipeline to the Amazon EC2 framework requires the development of some infrastructure and significant planning and testing. The pipeline was designed to operate in a traditional HPC cluster environment, hence we first ported the environment into EC2. Once that was completed, we began to decide where to locate our data, what size compute resources to use, and then conducted tests to validate our decisions.

3.1. Virtual cluster setup

The SNfactory code was developed to run on traditional HPC clusters. It assumes that a shared file system exists between the nodes, and that there is a head node that controls the distribution of work units. However, the Amazon EC2 environment only supports the creation of independent virtual machine instances that boot appropriate instances. To make it easier to port the SNfactory pipeline, we developed the ability to create virtual clusters in the EC2 environment. A virtual cluster connects a series of virtual machines together with a head node that is aware of all of the worker nodes, and a shared file system between the nodes.

To provide a persistent shared file system, we created an Amazon Elastic Block Storage (EBS) volume [3]. EBS provides a block level storage volume to EC2 instances that persists independently from the instance lifetimes. On top of the EBS volume we built a standard Linux ext3 file system. We were then able to have our head node export this file system via NFS to all of the virtual cluster nodes.

To setup a virtual cluster we tried two different techniques. The first technique we tried involved customizing the virtual machine images for each role. A custom image would be made that knew how to attach the EBS volume, start all of the worker nodes, and then export the EBS volume over NFS. While this approach had the advantage of simplicity for the end user, it quickly became apparent that it introduced a large burden on changing the environment. Any time a change was made, a new machine image had to be saved, and all of the other infrastructure updated to use this new image.

The process of creating and updating the images requires a fair understanding of Linux system administration and is tedious and time consuming. Thus, we decided to investigate alternative ways to handle the same setup. We use an EBS volume to store all application specific data and binary files. The application binary and data is then available on the workers through NFS. Also, it is necessary for the head node to know about the workers coming up and the workers need to know its master's physical address.

Thus, we decided to use a series of bash scripts that utilize the Amazon EC2 command-line tools, and completely automate the process of setting up the virtual cluster. All of the state is now kept in these scripts, and standard machine images can be used. During the setup of a virtual cluster, we first instantiate an instance that will become the head node of the virtual cluster, to this node we then attach the EBS volume. Once this is complete, we instantiate each of the worker nodes. For each worker node, we write its private IP address into the head nodes /etc/exports file. This file controls the addresses the NFS server will export to. The private IP addresses are also written out into a MPI machine file. The head node uses this file to decide where to send work units. After these files are written, the NFS server is started on the head node, and the proper mount commands are executed on the worker nodes. At this point in time, our virtual cluster setup is completed, and we are ready to begin running the SNfactory jobs.

3.2. *Data placement*

Once we had developed the mechanisms to create virtual clusters, we were faced with deciding where to store our input data, code, and output data. In a traditional cluster environment all data would be stored on the shared file system. In the Amazon Web Services environment we have two main choices for data storage. We can store data on the EBS volume that is shared between the nodes, or we can store our data in the Simple Storage Service (S3) [5]. S3 provides a simple web services interface to store and retrieve data from anywhere. To decide which of these options would provide the best performance at the cheapest cost, we ran a series of experiments that are described below.

4. Evaluation

The goal of our evaluation was to evaluate the various choices available to the end user through Amazon EC2 and study the impact on performance and corresponding cost considerations.

4.1. *Experimental setup*

We undertook a series of experiments focused on I/O and CPU data-processing throughput to observe and characterize performance, explore scalability, and discover optimal configuration strategies for the SNfactory using a virtual cluster in the Amazon Elastic Compute Cloud. We were particularly interested in studying the EBS versus S3 storage trade-offs, and the effects of various I/O patterns on aggregate performance for realistic "runs" of spectrograph data processing. In addition, we concentrated on approaches that required only minimal coupling between the existing SNfactory IFU pipeline and EC2 resources; for example, invasive changes that would enable the pipeline to access S3 were ruled out but transparent NFS file access was not.

Experiments were organized in a matrix by long-term storage option employed (EBS or S3), and whether or not I/O was concurrent with data processing or segregated into phases that preceded or followed it. Table 1 summarizes the options used for data placement in the experiments discussed below. Each experiment was first conducted using a cluster of 40 worker cores and repeated with 80 worker cores. For both experiments, each worker core was assigned one night of data. In each cluster, an additional node was allocated which attaches an EBS volume and serves it out to the workers via NFS. This configuration was used even when processing inputs and/or outputs were written to S3 – the NFS server was used to distribute the SNfactory pipeline's compiled binary executables and scripts to workers. We also address the implications of this strategy in our analysis.

EC2 32-bit high-CPU medium instances (c1.medium: 2 virtual cores, 2.5 EC2 Compute Units each) were used in all experiments discussed. Test runs with small instances (m1.small: 1 virtual core with 1 EC2 Compute Unit) demonstrated that a cluster consisting of those instances is actually less cost-effective by a factor of two since the cost per core is the same but the wall-clock time required for processing is twice as long: 30% of physical CPU resources are available to a single m1.small instance where nearly 95% are available to a single c1.medium instance. The relative cost ratio per core of 1:1 also holds in the Amazon EC2 spot-price market given the observed average spot-prices to date. However, it should be noted that this ratio is an ideal in the spot-market, where users declare a price above spot they are willing to pay.

For profiling our cluster, we found the *sysstat* project's *sar* command [19] to be a very useful way to collect information on system performance with very low overhead (sampling every 15 s results in a load of essentially 0.00 on an idle machine). The low overhead

Table 1
Experimental setup

Experiment	Input data	Output data
EBS-A1	EBS via NFS	Local storage to EBS via NFS
EBS-B1	Staged to local storage from EBS	Local storage to EBS via NFS
EBS-A2	EBS via NFS	EBS via NFS
EBS-B2	Staged to local storage from EBS	EBS via NFS
S3-A	EBS via NFS	Local storage to S3
S3-B	Staged to local storage from S3	Local storage to S3

was not surprising as *sar* reads kernel data structure values (i.e., counters) via the /proc file-system. There are about 200 quantities *sar* samples in "-A" mode, which we used excluding collection of interrupt statistics on all 256 interrupts. The *sar* utility was run on the NFS server and all worker nodes in each experiment, and its output served as the primary source for our measurements.

The raw spectrograph data set itself is organized (in general but also in particular on EBS) in a nested directory tree by night of observation – all files obtained in a given 24 h period are contained in a single directory (an average of about 370 files per directory). Data files themselves are mostly FITS [21] format, a standard digital image format widely used in astronomy. The input data used to perform our EC2 experiments consists of raw spectrograph CCD images including science frames (supernovae and spectroscopic standard stars) as well as associated instrument calibration frames (internal arc and continuum lamp exposures for flexure and response corrections). The average size of a typical night's worth of spectrograph data is 2.7 GB. Other raw data files from the spectrograph include metadata text files, and an entire stream of FITS files from the photometric imaging channel, which is currently handled using a separate pipeline. Nonetheless, operations tested consist of most of the numerically intensive portions of the SNfactory spectroscopic pipeline.

4.2. Experiment results

In our description of the experimental results, we focus on detailed results from the 80-core runs, and rely on the smaller 40-core runs to discuss scaling. The 80-core cluster is a reasonable approximation of the size of cluster the SNfactory expressed interest in using, as it puts within reach end-to-end processing of our entire multi-year data set (over 500 nights) on the timescale of a day or so. This ability is critical to the analysis, where differences in how early processing steps propagate down to changes in cosmological results.

4.2.1. Experiments EBS-A1 and EBS-B1

In experiment EBS-A1, the reduction pipeline running on each worker instance reads raw input files as they were needed directly from the EBS volume across NFS via the EC2 internal network. As automated data reduction proceeded, data output products (usually further FITS files) were deposited in worker local ephemeral storage. When all of the processing was complete on a worker, the output files were copied back to the EBS volume served by the NFS server node. Figure 1(a) provides a detailed view of the observed performance during EBS-A1.

Figure 1(a) displays the measured network send and receive rates, disk read and write rates, and system load for the NFS server node. The red dashed lines in the top two panels trace the rates of transfer of input data to the worker nodes from the EBS volume. The periodic, decaying spikes of activity are a natural side-effect of the homogeneity of the input data: each set of files is the same size and the amount of time to process each is highly correlated. Perturbations in run-times cause the peaks to decay and disk access to spread out. File caching accounts for the difference between the network send and disk read curves, induced by duplication of nights across some worker cores (duplication is not used to observe caching in all experiments). During the first 3 h of the experiment, CPU load on the NFS server (bottom panel) is negligible, but as workers complete tasks and begin sending data back, the network receive and disk write (black solid lines), and system load climb rapidly. Data rates of over 40 MB/s are achieved, and the NFS server load climbs to around 10. This phase lasts for over 4 h, longer than it took to process the data. The broad spike of data transfer to the NFS node just before 2 h into the experiment is the result of a "short" night of data – a worker core completed processing and was able to send its results back to the EBS volume before the rest of the nights were completed.

In Fig. 1(b), we show the profile of disk and network activity for a typical worker node in the cluster. Disk writes on the worker (raw files from the NFS server) occur at punctuated intervals as the pipeline completes individual tasks on each set of inputs and gets the next set. During the phase where outputs are sent back to the NFS server, we see that the worker is competing with other workers in the cluster for access, as 40 MB/s of bandwidth must be shared across 80 cores.

Experiment EBS-B1 repeated Experiment EBS-A1, except that raw input files were staged to worker ephemeral storage before processing began. In Fig. 2(a), we see that the NFS server achieves a very high rate of transfer to the workers – around 60 MB/s reading from the EBS volume, and 80 MB/s sending the data out to workers (again caching explains the difference). The long transfer phase back to EBS is again observed, as expected. Note in Fig. 2(b), one of the two cores of the worker node was responsible for a short night – it was able to send its results back to the NFS

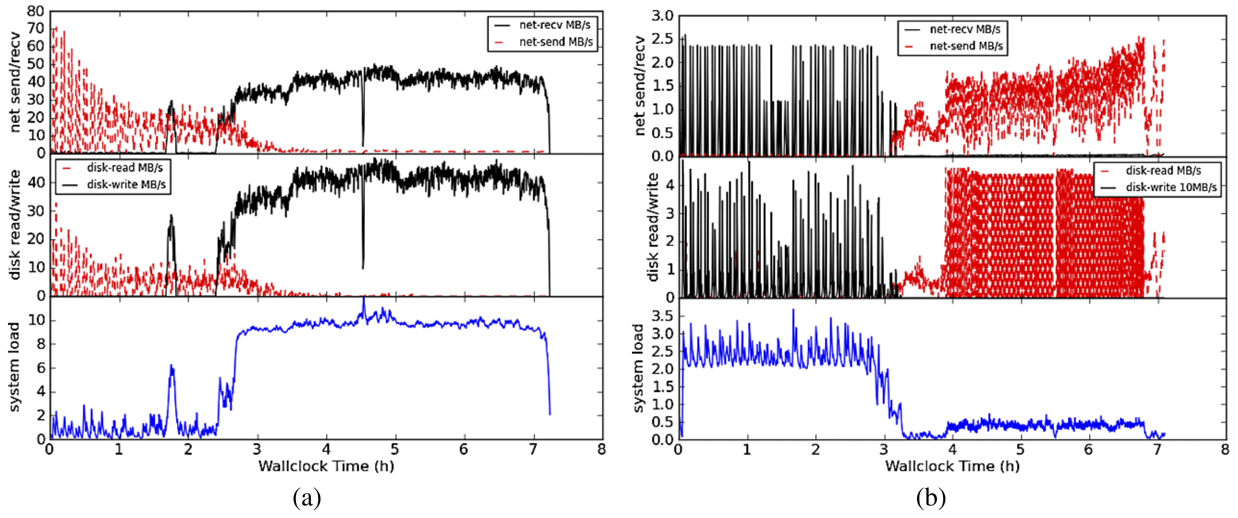


Fig. 1. Experiment EBS-A1. (a) NFS server. (b) Worker. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0324.)

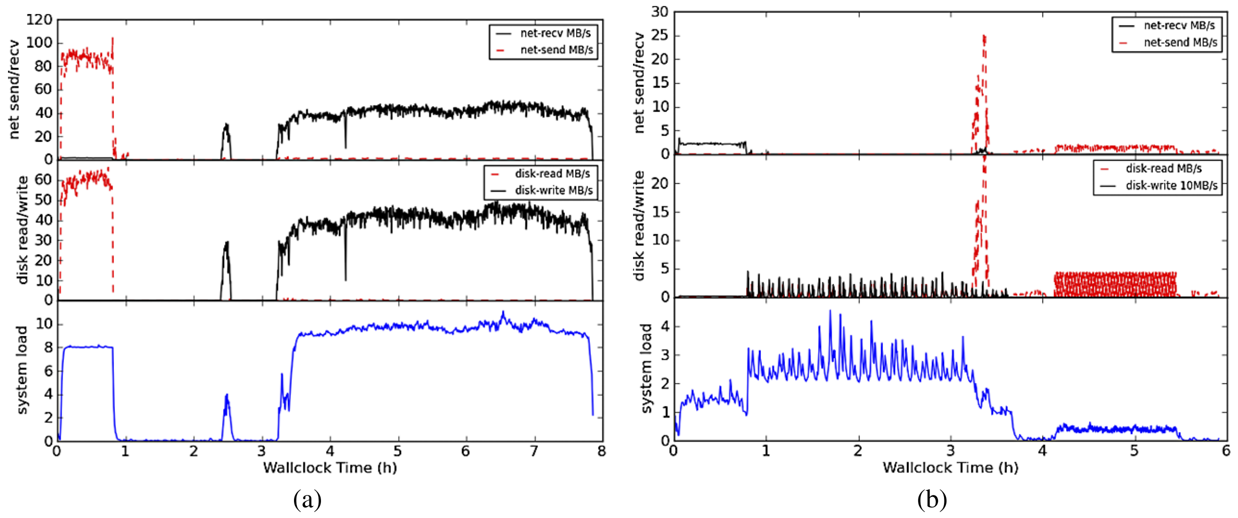


Fig. 2. Experiment EBS-B1. (a) NFS server. (b) Worker. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0324.)

server using all the bandwidth that is later shared by all 80 cores in the cluster, so here it enjoys superior network send and disk-read rates. The other night on the same node took longer to complete, and its output transfer lasted over an hour instead of a few minutes.

4.2.2. Experiments EBS-A2 and EBS-B2

For Experiment EBS-A2, the pipeline on each worker instance reads input files directly from EBS via NFS as in Experiment EBS-A1, but instead of caching the results and saving them to the EBS volume at the end of processing, they are saved back as they are produced. The point of the experiment is to determine whether interleaving the I/O with the data processing spreads out the I/O access patterns naturally to distribute bandwidth amongst worker cores.

Figure 3 shows this is simply not the case – in fact, a very strong oscillatory pattern in the data transfer rates and system load on the NFS server. We suspected that the stream of EBS writes to the NFS server was reducing the ability of workers to read the next set of inputs, driving a synchronization where tasks could not begin until all data had been sent to the EBS volume. Investigating the situation on the workers revealed something similar to this hypothesis but not exactly the same.

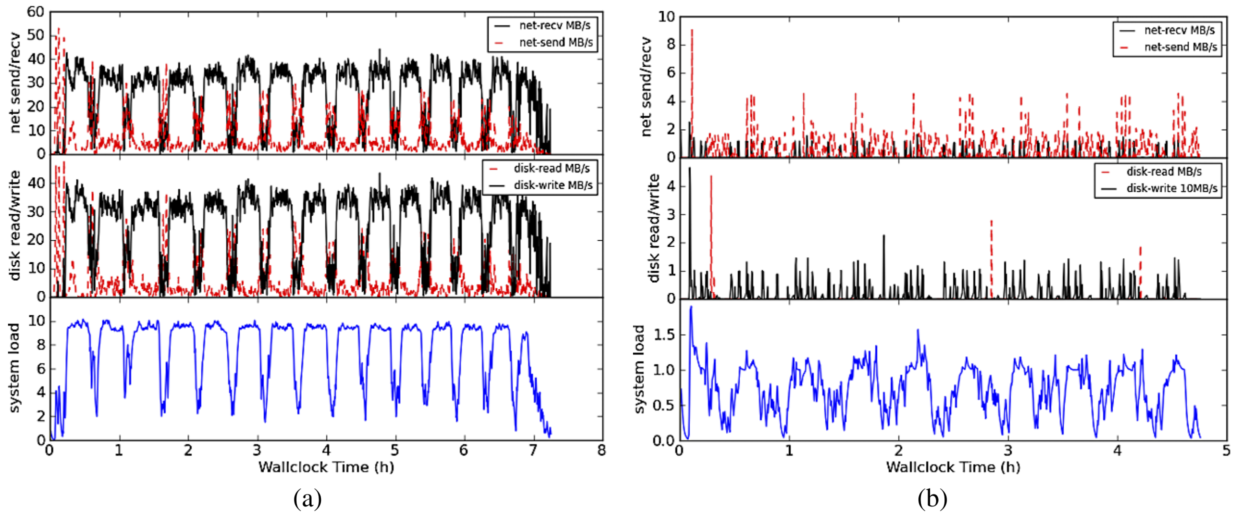


Fig. 3. Experiment EBS-A2. (a) NFS server. (b) Worker. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0324.)

Some SNfactory pipeline components appeared to be taking a very long time to complete appointed tasks, but did not seem to be utilizing the CPU heavily. Using the strace command, we found that in at least one case, the scripts were taking a very long time to simply load from the NFS server. In particular, a scattered-light correction script written in Python was observed to take 12 minutes simply to load as numerous modules in the form of shared objects, pure python, or compiled python were checked. Compiled binaries (say, written in C or C++) generally launched much faster than the interpreted scripts (which drove the synchronization).

Experiment EBS-B2 is a variation of EBS-A2, with raw input data being staged to worker local ephemeral storage before processing is launched. The same oscillatory pattern is observed as in EBS-A2, and the NFS server network send and disk read rates were comparable to those observed in experiment EBS-B1.

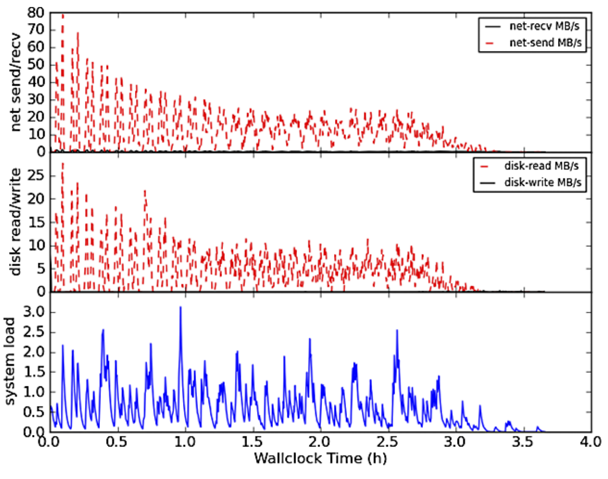
It is interesting to note that the 40-core runs did not exhibit the clear oscillatory behavior observed in the 80-core runs. This is true for both cases EBS-A2 and EBS-B2. Evidently the 40-core runs were below a threshold where the start-up times for interpreted scripts became noticeably enlarged.

4.2.3. *Experiments S3-A and S3-B*

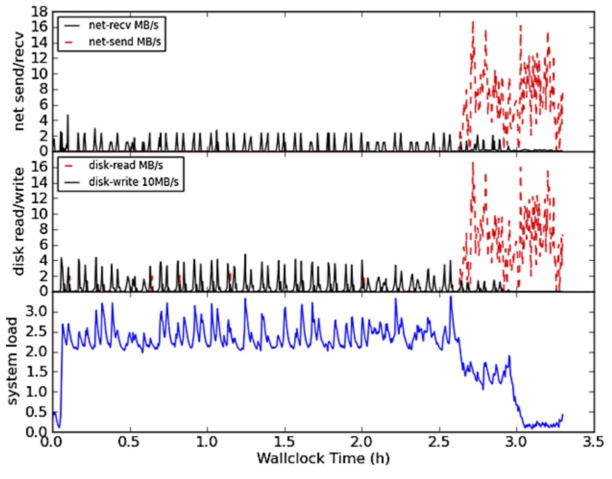
With the EBS-based experiments taking upwards of 7 h to complete, with at least as much time spent on file system access as was spent on CPU usage, we investigated using Amazon S3 as the persistent storage mechanism for pipeline inputs and outputs. In Experiment S3-A, raw input data was read from the EBS volume but the outputs cached to S3 after processing was done. Experiment S3-B relied upon S3 both to provide raw inputs staged to worker local ephemeral storage, and for long-term storage of outputs.

In Experiment S3-A, we again see the decaying network send and disk read rates on the NFS server in Fig. 4(a). As no outputs are being sent back to the EBS volume there is no measured net receive or disk write rate. On the worker, as depicted in Fig. 4(b), processing for one of the two nights completes about a half hour before the other and its transfer to S3 begins (and the load drops by 1). All data products are sent to S3, as observed by *sar* within much less than an hour. By using S3 as the destination of the output products of the pipeline, each worker apparently is able to achieve transfer rates of upwards of 6–8 MB/s, greater than when the workers share a single EBS volume over NFS. Note that S3's latency may mean that new data products sent to S3 may not be accessible by another node in the cluster or cloud immediately, but this is not a major concern. Whether or not the files were sent to S3 one at a time, or as a block, made no significant difference.

Experiment S3-B merely aggregated the staging of raw inputs to the front of the experiment and much the same behavior was observed as in S3-A on the worker nodes. The EBS volume was still accessed by workers in order to obtain scripts and binary executables needed to perform processing operations, however. But, the data transfer rates to support this across NFS (when no other appreciable traffic is present) results in no noticeable anomalous slow start-ups for interpreted scripts.



(a)



(b)

Fig. 4. Experiment S3-A. (a) NFS server. (b) Worker. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0324.)

In the 80-core experiments, the S3 variants clearly outstrip the EBS variants in terms of performance. Where a run of processing took around 7 h for the EBS variants, only 3 h were used in the S3 experiments. The amount of time spent by workers loading output data into S3 was an order of magnitude smaller than into EBS. Possible EBS-based solutions that could improve EBS performance include – splitting the data across multiple EBS volumes, or creating a RAID 0 array from multiple EBS volumes. However, these improvements, unless overwhelmingly cost-effective, would not be of interest to SNfactory due to increased complexity.

4.2.4. *Scaling*

Figure 5 compares mean wallclock times for each of the three main phases of each processing experiment. For comparison, the 40-core (21 nodes: 20 worker nodes and 1 NFS/head node) variants are included along side the 80-core (41 nodes: 40 worker nodes and 1 NFS/head node) results. The “fetch” phase is measured only for the “B” experiments that have a separate initial staging phase. In the other experiments, the file transfer from NFS to workers is combined with the processing (or “IFU” phase). The “put” phase is measured when outputs are sent to long-term storage from workers after processing is done. The wallclock time measurements are a mean over all workers in the cluster, and the distribution is dominated by the spread over the size of each input task, not conditions in EC2.

The scaling results in this figure are interesting. The S3 experiments scaling performance from 40 to 80-cores, which are reasonable sizes of interest to the

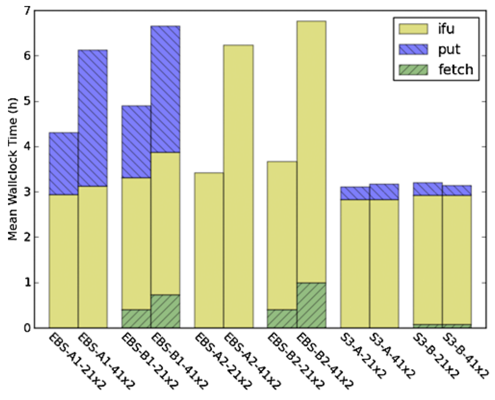
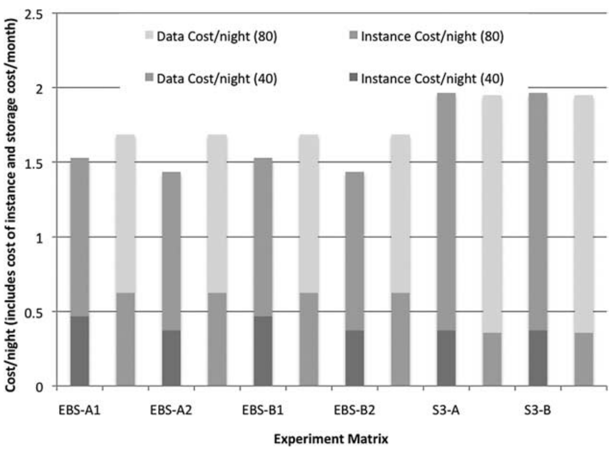


Fig. 5. Scaling performance. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0324.)

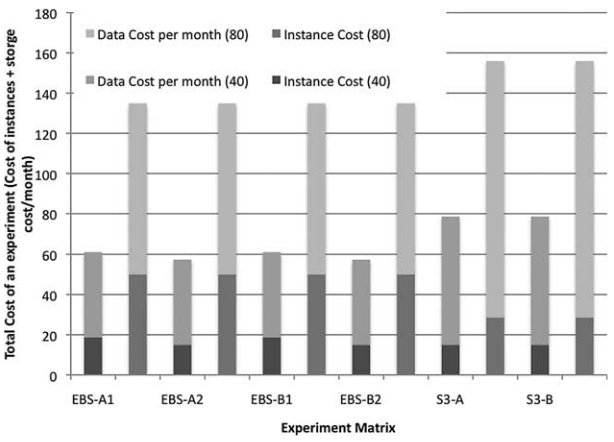
SNfactory, is excellent. Comparing the EBS-results, going from the “A” mode of putting outputs on the EBS volume after all processing to the “B” mode of interleaving transfer back resulted in a decrease of total time to complete the experiment, but this clearly did not translate from 40 cores to 80, where basically no change is observed.

4.2.5. *Cost*

We now attempt to calculate the rough cost of running the SNfactory pipeline in the AWS environment in order to understand if it would be cost effective for production usage. Figure 6(a) shows the cost, calculated using February 2010 prices, associated with analyzing one night of data with both 40 and 80 cores. Each bar represents the total costs. The data costs and compute costs are shown in different shades for each



(a)



(b)

Fig. 6. Costs. (a) Cost of analyzing one night of data. (b) Cost per experiment.

bar. We can clearly see that although S3 offers significantly better performance, that performance comes at a cost. Data storage in S3 is more expensive then storing data in an EBS volume.

Figure 6(b) shows the cost of running a single experiment and storing the data of that single experiment for one month. More realistic and useful research and development tasks, such as signal extraction tests on Monte Carlo simulations of data cubes, generally tend to be around the same size as the experiments conducted in this study. Data transfers between EC2 and S3 are free. Because of the cost numbers, it is clear that we should only use S3 storage for those places where it impacts performance. Otherwise we are better off using EBS for our storage. In the case of the SNfactory, this means that we will store our input data, and our application data in EBS. Our output data will be written to S3.

To date, the SNfactory experiment has around 700 nights of on-sky SNIFS data to process. Adopting a round figure of \$1.50 to process one night of data (based on Fig. 6(a)), we find that the total cost to reduce all the data is around \$1K. This assumes that files created during the data reduction procedure, including ancillary by-products, would be stored on S3 for a month. In practice, SNfactory tends to retain around 3–4 productions on disk for comparison, and does not reprocess all of its raw data every time (e.g., initial preprocessing is done once and subsequent productions can re-use those data products as a starting point). This complicates the cost estimate, but an upper bound assuming all the data were re-processed each time and kept on disk for up to 4 months would raise the cost per production to \$50K. In view of this potentially very large cost, SNfactory would have to devise alternate schemes or prioritize its use of disk space, for example, only re-running the disk-space heavy pre-processing when absolutely necessary.

Table 2

Microbenchmark experimental setup

Experiment name	Storage name
EC2-Local	Local storage (instance root device)
EC2-EBS	EBS storage (instance root device)
EC2-EBS-ext3	EBS storage (EBS ext3 mount on instance)

5. Microbenchmarks

To further understand the SNfactory performance on Amazon AWS, we conducted a series of microbenchmarks designed to measure the performance of the underlying storage systems.

5.1. Experimental setup

We performed a series of experiments to measure the file system latency and the file system read/write characteristics of Amazon EC2 instances when using both the local ephemeral storage and the Amazon EBS storage. In all the experiments we used a 32-bit CentOS image running on the high-CPU c1.medium instance type. Table 2 shows the various configurations of local and EBS storage options used for running the experiments.

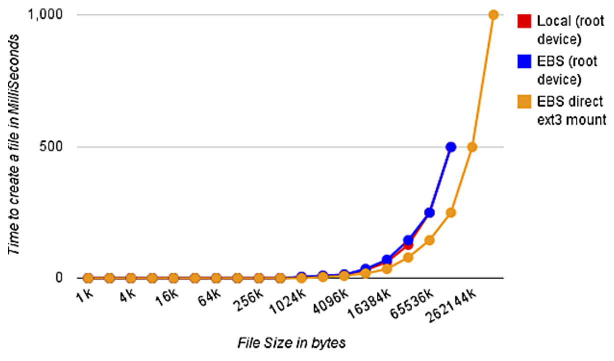
We used a freely distributed GPL micro-benchmark suite, called *lmbench* [15], to measure the file system latencies, and the HPC standard IOR [20] benchmark for measuring read/write throughputs. *lmbench* measures both file creation and file deletion latencies for varying file sizes and many of the benchmark features like number of test repetitions, file system selection to perform creation/deletion tests are easily configurable. We measured file creation and deletion latencies for file sizes ranging from 1 KB to 512 MB. To measure read/write throughput, IOR is configured to perform single shared file POSIX I/O to the filesystem under test. Each of the CPUs in the two CPU c1.medium instance is made to write about 1 GB of data to the file in chunks of 1 MB of data in each data transfer.

5.2. *Experimental results*

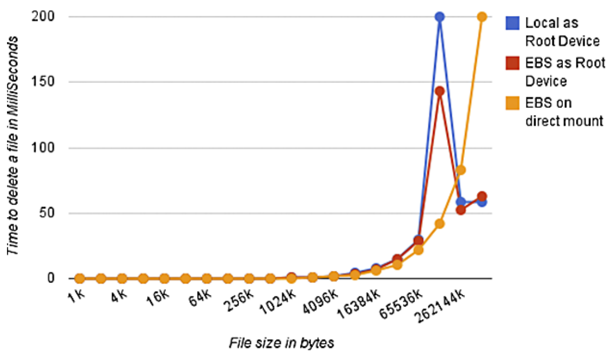
Figure 7(a) shows the file creation latencies as measured by *lmbench*. We can see that file creation latencies remain close to zero for file sizes up to 512 KB. For file sizes after 512 KB the latency increases exponentially. While all storage options show similar latencies, EBS over ext3 mount provides the best latency, especially as file sizes increase.

Similar to creation, Fig. 7(b) file deletion latencies remain close to zero for all cases until sizes of 8 MB are reached. Latencies continue to increase as file sizes increase, however, for both of the root devices, latencies suddenly drop at 256 MB. We have not been able to identify a cause for this drop. For the file sizes used by SNfactory, it is clear from both latency benchmarks that file creation and file deletion performance is adequate to meet SNfactory needs.

Figure 8(a) shows the read throughput in MB/s as measured with IOR. The read performance of both the EBS root device and the mounted EBS volume are similar. The local, non-EBS, root devices shows significantly poorer performance.

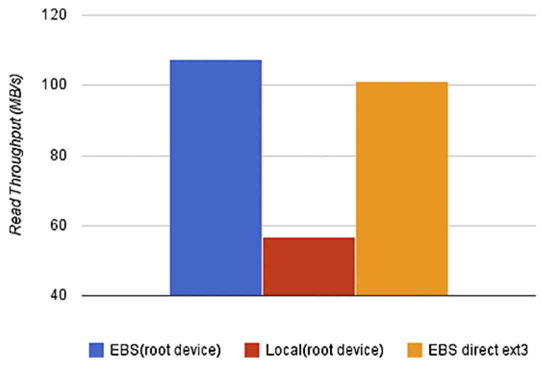


(a)

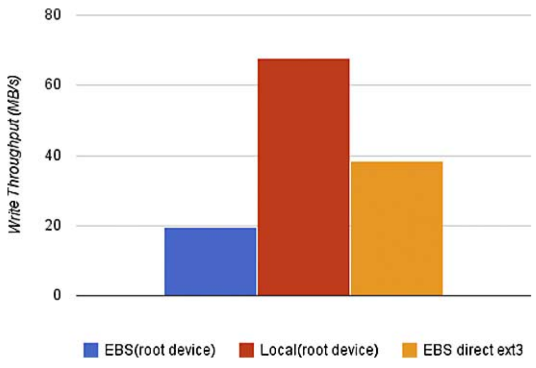


(b)

Fig. 7. lmbench results. (a) File creation. (b) File deletion. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0324.)



(a)



(b)

Fig. 8. IOR results. (a) Read throughput. (b) Write throughput. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0324.)

Figure 8(b) shows the write throughput in MB/s as measured with IOR. This graph clearly shows the underlying performance bottleneck that constrained the SNfactory performance when writing output data to EBS. The observed write performance in the IOR tests is less then 40 MB/s. This is less then the performance achieved via NFS in earlier testing, but we believe the difference is due to variations in the AWS load and infrastructure. The IOR tests were conducted 9 months after the original tests were conducted.

6. Related work

Various application groups have run scientific computations and pipelines on Amazon EC2 to study the feasibility of the model for a particular application. In addition previous work has evaluated performance of difference Amazon components, e.g., the storage service (S3) [5].

Deelman et al. detail the computational and storage costs of running the Montage workow on Amazon EC2 resources [10]. High-Energy and Nuclear Physics (HEPN)'s STAR experiments have been run on cloud environments such as Amazon EC2 [13,14] and identified certain challenges associated with image management, deployment context and the management of virtual clusters. Standard benchmarks have been evaluated in the Amazon EC2 environment that experience communication bottlenecks but small-scale codes [11].

Our work details both the development as well as the performance impact of various design solutions when running the SNfactory Experiment on Amazon EC2.

7. Conclusion

While the Amazon Web Services environment can be very useful for scientific computing, porting your scientific application into this framework today requires significant effort. Most scientific applications expect a certain environment to be present. Either that environment, e.g., an HPC cluster environment, is replicated in EC2, or the application must be changed to remove those expectations.

One expectation that most scientific applications that run in traditional cluster environments have is that the mean rate of failure is very low. Traditionally this has been true; hence most scientific applications do not handle failure well. Our experience with the Amazon Web Services environment is that failures occur frequently, and the application must be able to handle them gracefully and continue operation.

The most common failure is an inability to acquire all of the virtual machine images you requested because insufficient resources are available. When attempting to allocate 80 cores at once, this happens fairly frequently. Your application needs to be able to adapt to the actual number of virtual machines available, and not expect that it will always acquire all of the requested resources.

In addition to not being able to acquire all of the requested resources, we saw a wide variety of transient errors. These included an inability to access the âuser-dataâ passed in during image startup, failure to properly configure the network, failure to boot properly, and other performance perturbations. While none of these errors occurred frequently, they do in aggregate happen often enough that it is essential that your application can deal gracefully with them.

Virtualized environments are attractive to scientific users since it gives users the flexibility to run custom software environments specific to a project. This flexibility however comes at a cost. Determining the Amazon cloud services to use and setting up the virtual cluster configuration for the workflow is non-trivial. In addition, creating virtual images requires a modest understanding of Linux system administration. Thus, porting and running an application in the Amazon cloud environment requires a good understanding of the services offered, modest amount of time, middleware pieces to manage the orchestration of the various services and specialized skills for managing images.

In addition to managing errors, and essential component of porting a scientific application into the Amazon Web Services environment is benchmarking. Understanding how to utilize the various storage components available in the environment today to maximize performance for a given cost requires a significant effort in benchmarking your application.

From our experiments, we conclude that at least for the SNfactory, an optimal configuration of Amazon Web Services resources consists of storing raw input files on an EBS volume, and sending the outputs to Amazon S3, capitalizing on S3's superior scaling properties. Application code can reside on an EBS volume shared out over NFS.

Acknowledgements

This work was funded in part by the Advanced Scientific Computing Research (ASCR) in the DOE

Office of Science under contract number DE-C02-05CH11231. The authors would like to thank Amazon for access to Amazon EC2. The authors would also like to thank the Magellan team at NERSC for discussions on cloud computing.

References

[1] G. Aldering, G. Adam, P. Antilogus, P. Astier, R. Bacon, S. Bongard, C. Bonnaud, Y. Copin, D. Hardin, F. Henault, D.A. Howell, J. Lemonnier, J. Levy, S.C. Loken, P.E. Nugent, R. Pain, A. Pecontal, E. Pecontal, S. Perlmutter, R.M. Quimby, K. Schahmaneche, G. Smadja and W.M. Wood-Vasey, Overview of the Nearby Supernova Factory, in: *The Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, J.A. Tyson and S. Wolff, eds, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 4836, SPIE, Bellingham, WA, 2002, pp. 61–72.

[2] Amazon Web Services, http://aws.amazon.com/.

[3] Amazon EBS, http://aws.amazon.com/ebs/.

[4] Amazon EC2, http://aws.amazon.com/ec2/.

[5] Amazon S3, http://aws.amazon.com/ebs/s3/.

[6] E. Angerson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof and D. Sorensen, LAPACK: a portable linear algebra library for high-performance computers, in: *Proceedings of Supercomputing'90*, IEEE, New York, NY, 2002, pp. 2–11.

[7] C. Aragon, S. Poon, G. Aldering, R. Thomas and R. Quimby, Using visual analytics to develop situation awareness in astrophysics, *Information Visualization* **8**(1) (2009), 30–41.

[8] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica et al., Above the clouds: a Berkeley view of cloud computing, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, CA, 2009.

[9] CFITSIO, http://heasarc.nasa.gov/docs/software/fitsio/fitsio.html.

[10] E. Deelman, G. Singh, M. Livny, B. Berriman and J. Good, The cost of doing science on the cloud: the montage example, in: *Proceedings of SC'08*, IEEE, Austin, TX, 2008, pp. 1–16.

[11] 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, in: *Proceedings of the Cloud Computing and Its Applications*, Chicago, IL, October 2008.

[12] GSL, http://www.gnu.org/software/gsl/.

[13] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman and M. Tsugawa, Science clouds: Early experiences in cloud computing for scientific applications, in: *Proceedings of the Cloud Computing and Applications*, Chicago, IL, 2008.

[14] K. Keahey, T. Freeman, J. Lauret and D. Olson, Virtual workspaces for scientific applications, *Journal of Physics: Conference Series* **78** (2007), 012038.

[15] L. McVoy and C. Staelin, lmbench: portable tools for performance analysis, in: *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, Usenix Association, Berkeley, CA, 1996, p. 23.

[16] M. Palankar, A. Iamnitchi, M. Ripeanu and S. Garfinkel, Amazon S3 for science grids: a viable solution?, in: *Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing*, ACM, New York, NY, 2008, pp. 55–64.

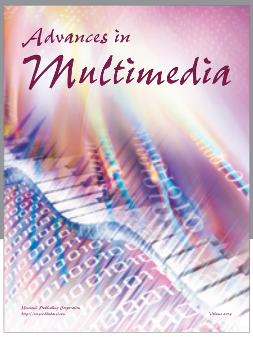
[17] S. Perlmutter, G. Aldering, G. Goldhaber, R. Knop, P. Nugent, P. Castro, S. Deustua, S. Fabbro, A. Goobar, D. Groom et al., Measurements of â and Λ from 42 high-redshift supernovae. *The Astrophysical Journal* **517** (1999), 565–586.

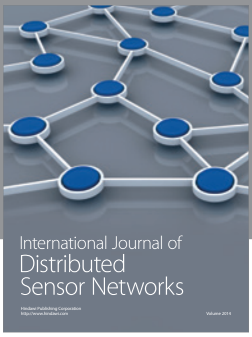
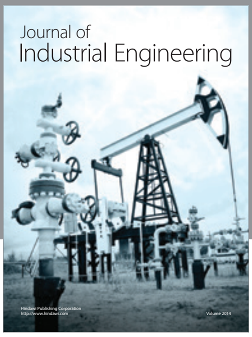
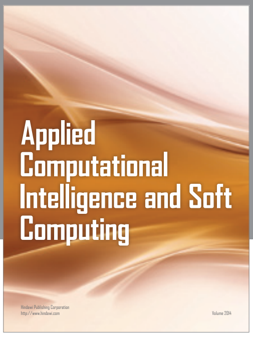
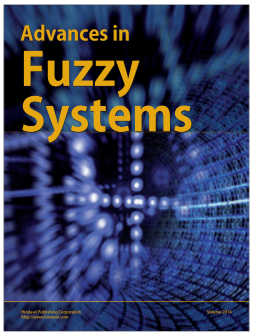
[18] A. Riess, A. Filippenko, P. Challis, A. Clocchiatti, A. Diercks, P. Garnavich, R. Gilliland, C. Hogan, S. Jha, R. Kirshner et al., Observational evidence from supernovae for an accelerating universe and a cosmological constant, *The Astronomical Journal* **116** (1998), 1009–1038.

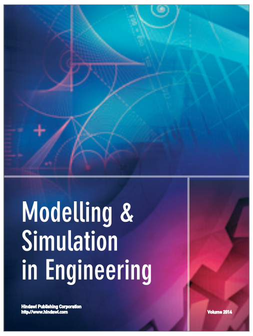
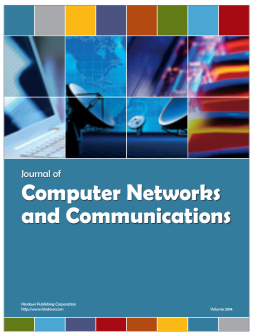
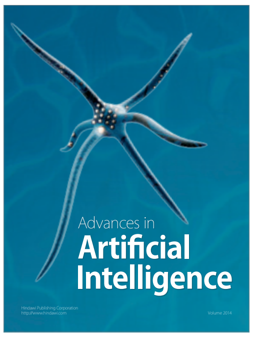
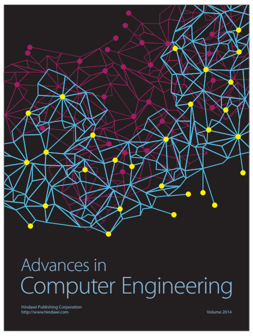
[19] Sar, http://pagesperso-orange.fr/sebastien.godard/sw.

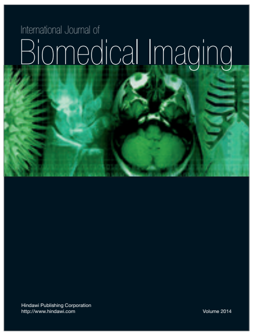
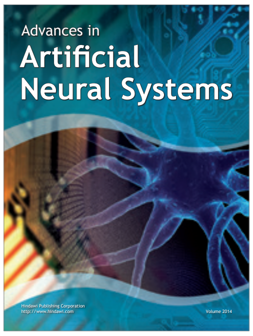
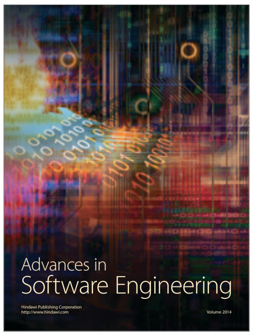
[20] H. Shan and J. Shalf, Using ior to analyze the i/o performance for HPC platforms, in: *Cray User Group Conference*, Seattle, WA, May 7–10, 2007.

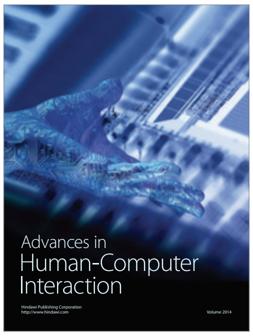
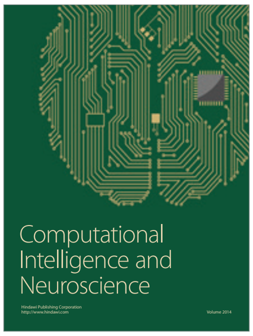
[21] D. Wells, E. Greisen and R. Harten, FITS-a flexible image transport system, *Astronomy and Astrophysics Supplement Series* **44** (1981), 363–370.


Advances in
Multimedia

The Scientific World Journal

International Journal of
Distributed Sensor Networks

Journal of
Industrial Engineering

Applied Computational Intelligence and Soft Computing

Advances in
Fuzzy Systems


Modelling & Simulation in Engineering

Journal of
Computer Networks and Communications

Advances in
Artificial Intelligence

Advances in
Computer Engineering

International Journal of
Computer Games Technology

International Journal of
Biomedical Imaging

Advances in
Artificial Neural Systems

Advances in
Software Engineering

Journal of
Robotics

Advances in
Human-Computer Interaction

Computational Intelligence and Neuroscience

International Journal of
Reconfigurable Computing

Journal of
Electrical and Computer Engineering