

# Seeking Supernovae in the Clouds: A Performance Study

Keith R. Jackson  
Lavanya Ramakrishnan  
Computational Research Division  
Lawrence Berkeley National Lab  
1 Cyclotron Road  
Berkeley, CA 94720  
KRJackson@lbl.gov  
LRamakrishnan@lbl.gov

Karl J. Runge  
Physics Division  
Lawrence Berkeley National Lab  
1 Cyclotron Road  
Berkeley, CA 94720  
(510) 486-4384  
KJRunge@lbl.gov

Rollin C. Thomas  
Computational Cosmology Center  
Lawrence Berkeley National Lab  
1 Cyclotron Road  
Berkeley, CA 94720  
(510) 486-4697  
RCThomas@lbl.gov

## 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. As part of its effort 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 ~10TBs of data.

This pipeline has traditionally been run on a local cluster. Cloud computing 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.

## Categories and Subject Descriptors

D.1.3 [Programming Techniques]: Concurrent programming – Distributed programming.

## General Terms

Performance, Design.

## Keywords

Cloud Computing, Distributed Systems, High Performance Computing, eScience.

## 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 the search space for their application.

Cloud computing [20] 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 bit 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 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 scientific user communities. Earlier studies have evaluated performance implications for applications on cloud environments and experimented with specific choices of deployment [12-16]. 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) [4] 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.

## 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 at 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 [9].

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 [11]. The entire process, from the start of data collection to human identification of candidates, took approximately 18 hours.

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 are 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, followed-up 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. This data 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 [6], the GNU Scientific Library (GSL) [7] and Python libraries like scipy and numpy (which in turn also depend on BLAS [5], LAPACK [10], 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) [2]. Key aspects that were initially attractive to SNfactory include:

- 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 first we began 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 boots 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 [1]. 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.

Instead of this approach, we decided to use a series of bash scripts that utilize the Amazon EC command-line tools. 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 work 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) [3]. 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 with long-term storage was concurrent with data processing or segregated into phases that precede or follow it. Each experiment was first conducted using a cluster of 40 worker cores and repeated with 80 worker cores. 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 involved 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 [8] to be a very useful way to collect information on system performance with very low overhead (sampling every 15 seconds results in a load of essentially 0.00 on an idle machine).

The low overhead 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-hour period are contained in a single directory (an average of about 370 files per directory). Data files themselves are mostly FITS [19] 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.

Table 1 Experimental Data Placement

| 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          |

### 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 their 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.

Table 1 summarizes the options used for data placement in the experiments discussed below.

#### 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 provides a detailed view of the observed performance during EBS-A1.

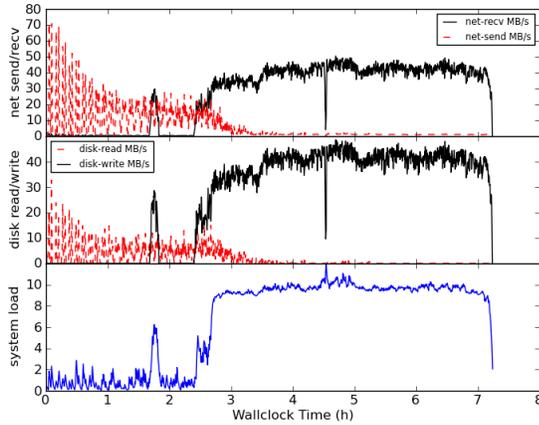


Figure 1 Experiment EBS-A1 NFS Server

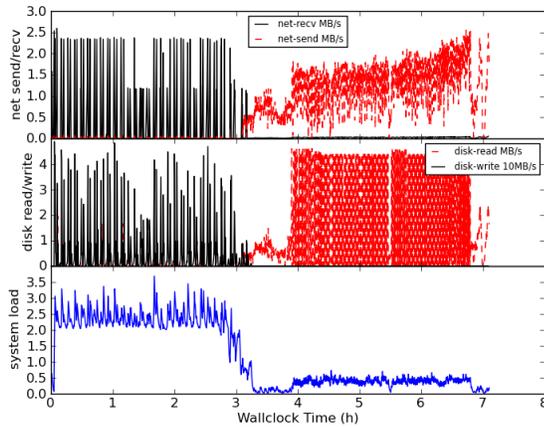


Figure 2 Experiment EBS-A1 Worker

Figure 1 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 hours 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 hours, longer than it took to process the data. The broad spike of data transfer to the NFS node just before

2h 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 Figure 2, 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.

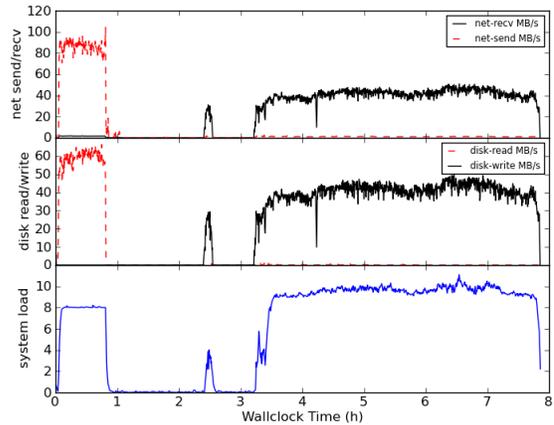


Figure 3 Experiment EBS-B1 NFS Server

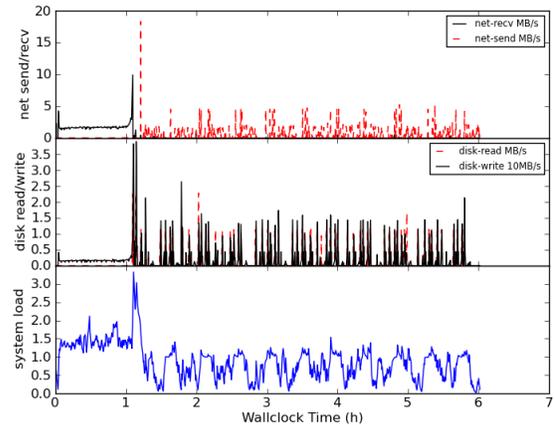


Figure 4 Experiment EBS-B1 Worker

Experiment EBS-B1 repeated Experiment EBS-A1, except that raw input files were staged to worker ephemeral storage before processing began. In Figure 3, 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 Figure 4, 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 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.

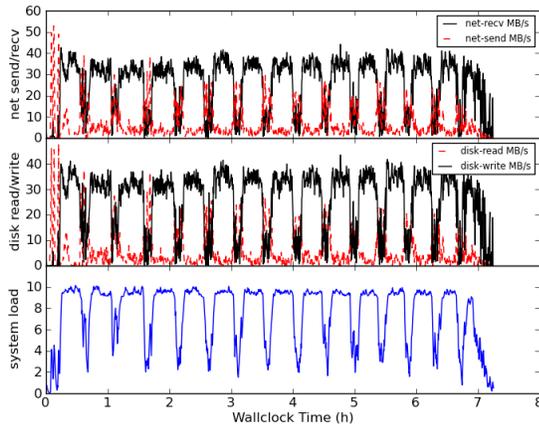


Figure 5 Experiment EBS-A2 NFS Server

Figures 5 and 6 show 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.

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.

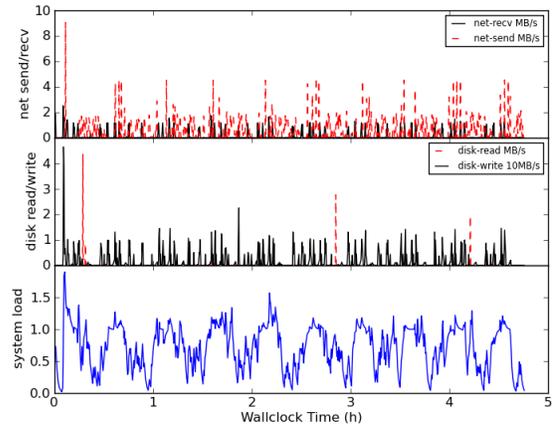


Figure 6 Experiment EBS-A2 Worker

#### 4.2.3 Experiments S3-A and S3-B

With the EBS-based experiments taking upwards of 7 hours 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.

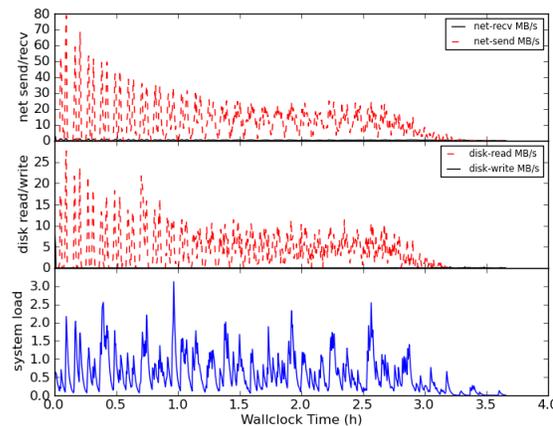
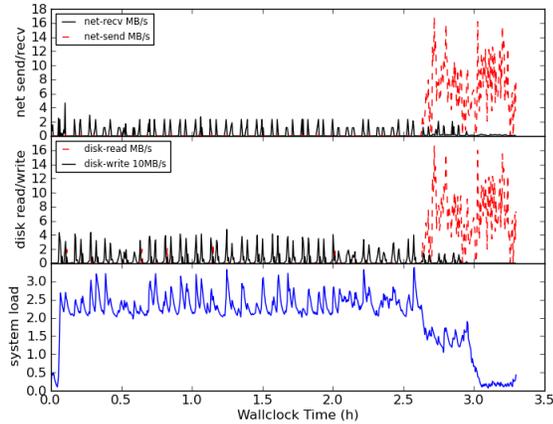


Figure 7 Experiment S3-A NFS Server

In Experiment S3-A, we again see the decaying network send and disk read rates on the NFS server in Figure 7. 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 Figure 8, 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.



**Figure 8 Experiment S3-A Worker**

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.

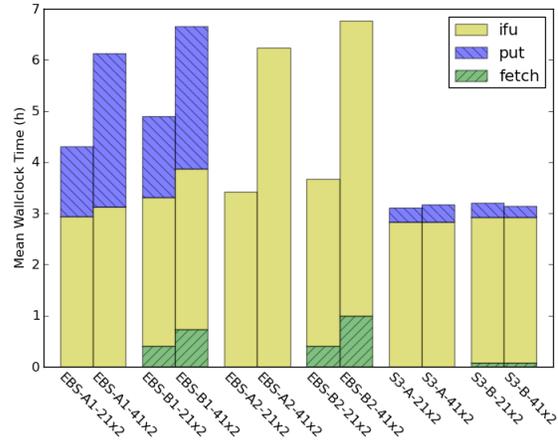
In the 80-core experiments, the S3 variants clearly outstrip the EBS variants in terms of performance. Where a run of processing took around 7 hours for the EBS variants, only 3 hours were used in the S3 experiments. The amount of time spent by workers loading output data into S3 was an order of magnitude 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 9 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

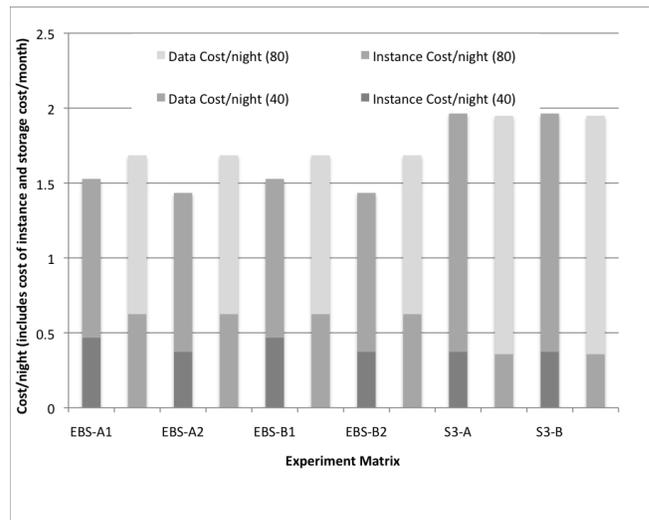
reasonable sizes of interest to the SNfactory, is excellent. Comparing the EBS-results, going from the "A" mode of putting outputs on the EBS volume after all processing to the "B" mode of interleaving transfer back resulted in a decrease of total time to complete the experiment, but this clearly did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



**Figure 9 Scaling Performance**

#### 4.2.5 Cost

Figure 10 shows the cost 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 show in different shades for each 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 10 Cost of Analyzing One Night of Data**

Figure 11 shows the cost of running a single experiment and storing the data of that single experiment for one month. Data transfers between EC2 and S3 are free. Because of the cost numbers, it is clear that we should only use S3 storage in 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.

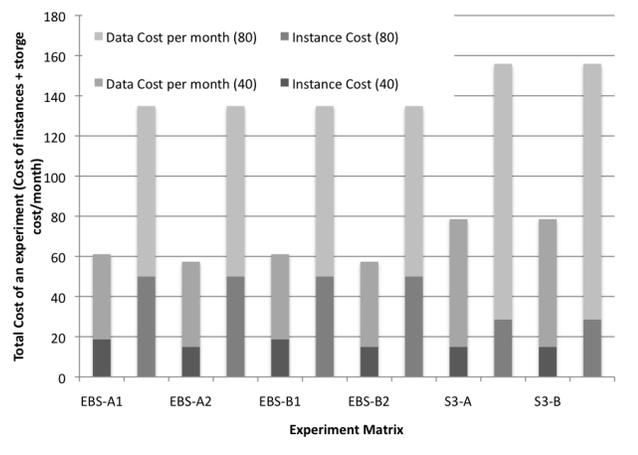


Figure 11 Cost per Experiment

## 5. 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) [16].

Deelman et al. detail the computational and storage costs of running the Montage workflow on Amazon EC2 resources [12]. High-Energy and Nuclear Physics (HEPN) ‘s STAR experiments have been run on cloud environments such as Amazon EC2 [14, 15] 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 [13].

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

## 6. 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 “user-data” 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.

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.

## 7. ACKNOWLEDGMENTS

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.

## 8. REFERENCES

- [1] Amazon EBS. <http://aws.amazon.com/ebs/>
- [2] Amazon EC2. <http://aws.amazon.com/ec2/>
- [3] Amazon S3. <http://aws.amazon.com/s3/>
- [4] Amazon Web Services. <http://aws.amazon.com/>
- [5] BLAS. <http://www.netlib.org/blas/>
- [6] CFITSIO. <http://heasarc.nasa.gov/docs/software/fitsio/fitsio.html>
- [7] GSL. <http://www.gnu.org/software/gsl/>
- [8] Sar. <http://pagesperso-orange.fr/sebastien.godard/>
- [9] Aldering, G. et al., *Overview of the Nearby Supernova Factory*. City, 2002.
- [10] Anderson, E., Bai, Z., Dongarra, J., Greenbaum, A., McKenney, A., Du Croz, J., Hammerling, S., Demmel, J., Bischof, C. and Sorensen, D. *LAPACK: A portable linear algebra library for high-performance computers*. IEEE Computer Society Press, City, 1990.
- [11] Aragon, C. R., Poon, S. S., Aldering, G. S., Thomas, R. C. and Quimby, R. Using visual analytics to develop situation awareness in astrophysics. *Information Visualization*, 8, 1 (2009), 30-41.
- [12] Deelman, E., Singh, G., Livny, M., Berriman, B. and Good, J. *The Cost of Doing Science on the Cloud: The Montage Example*. City, 2008.
- [13] Evangelinos, C. and Hill, C. N. *Cloud computing for Parallel Scientific HPC Applications: Feasibility of Running*

*Coupled Atmosphere-Ocean Climate Models on Amazon's EC2*. City, 2008.

[14] Keahey, K. and Freeman, T. *Science Clouds: Early Experiences in Cloud Computing for Scientific Applications*. City, 2008.

[15] Keahey, K., Freeman, T., Lauret, J. and Olson, D. Virtual Workspaces for Scientific Applications. *Physics: Conference Series*, 78, 012038 (2007 2007), 5.

[16] Palankar, M. R., Iamnitchi, A., Ripeanu, M. and Garfinkel, S. *Amazon S3 for science grids: a viable solution?*, City, 2008.

[17] Perlmutter, S., et al. Measurements of Omega and Lambda from 42 High-Redshift Supernovae. *Astrophys. J.*, 517, (1999), 565-586.

[18] Riess, A., et al., Observational Evidence from Supernovae for an Accelerating Universe and a Cosmological Constant, *Astron. J.*, 116, (1998), 1009-1038.

[19] Wells, D. C., Greisen, E. and Harten, R. H. FITS: A Flexible Image Transport System. *Astron. Astrophys. Suppl.*, 44, (1981), 363-370.

[20] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A. and Stoica, I. Above the clouds: A berkeley view of cloud computing. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-282009).