Scientific computing using virtual high-performance computing: a case study using the Amazon Elastic Computing Cloud

Scott Hazelhurst
School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg
Private Bag 3, 2050 Wits, South Africa
Scott.Hazelhurst@wits.ac.za

ABSTRACT

High-performance computing systems are important in scientific computing. Clusters of computer systems — which range greatly in size — are a common architecture for high-performance computing. Small, dedicated clusters are affordable and cost-effective, but may not be powerful enough for real applications. Larger dedicated systems are expensive in absolute terms and may be inefficient because many individual groups may not be able to provide sustained workload for the cluster. Shared systems are cost-effective, but then availability and access become a problem.

An alternative model is that of a virtual cluster, as exemplified by Amazon's Elastic Computing Cloud (EC2). This provides customers with storage and CPU power on an on-demand basis, and allows a researcher to dynamically build their own, dedicated cluster of computers when they need it. Used by commercial web services deployers, this technology can be used in scientific computing applications. This paper presents a case study of the use of EC2 for scientific computing. The case study concludes that EC2 provides a feasible, cost-effective model in many application areas.

Categories and Subject Descriptors

C2.4 [**Computer communication systems**]: Distributed Systems; D4.0 [**Operating systems**]: Distributed systems; J.0 [**Computer applications**]: General

General Terms

Algorithms, Performance, Measurement

Keywords

High-performance computing, clusters, virtualisation, Amazon Elastic Computing Cloud

1. INTRODUCTION

Many scientific and other applications require high-performance computing — a catch-all phrase to describe applications characterised by large data sets requiring significant computing resources. Such applications may require hundreds to hundreds of thousands of CPU-hours, and so to be practical have to be parallelised.

A popular architecture for parallelisation is a cluster of commodity computers, connected by a fast network. This encompasses a range of systems. At the low-end are low-power or mid-range computers connected by ethernet (100 MB or 1GB rates), perhaps with no special-purpose switch. At the higher-end, the individual CPUs are more powerful, with a high-performance and very expensive interconnect. As an example, the iQudu cluster of the South African Centre for High Performance Computing (CHPC), consists of 160 nodes. Each node has 16G of RAM and contains two CPUs, dual-core AMD Opterons 2218s (so there are 640 cores in total in the system). The cluster is connected both with gigabit ethernet and an Infiniband system.

Clusters thus range in performance, and of course price. For researchers wishing to use a cluster there are a range of options:

- A small dedicated cluster. This system is often affordable by a group or individual researcher, and is reasonably cost-effective. The computers in the system may have other uses – for example personal workstations or teaching equipment – but are under the direct control of the group who can get dedicated access to them. However, for some scientific applications small clusters are insufficient in size.
- Large, dedicated clusters. These are desirable, but absolute cost becomes prohibitive. Auxiliary costs such as air-conditioning, power protection and security become much more significant. Moreover, especially in research environments, these type of systems may not be cost-effective as few groups have applications that require sustained very large-scale application use.
- Large, shared clusters such as the iQudu cluster at the CHPC (`www.chpc.ac.za`) or the C4 cluster hosted at the African Advanced Institute for Information &

Communication Technology (also known as the Meraka Institute, `www.meraka.org.za`). Since these facilities are shared, high-performance systems are affordable and are cost-effective as several groups sharing the facility can ensure relatively high utilisation. The disadvantage from a researcher's perspective is that access to the equipment is mediated by technical and social constraints. Usually, these systems can only be accessed through a job-scheduling system such as Oscar or LoadLeveler, which is not ideal for many research applications. The equipment is shared (Sod's law implies that at the point you need to run a big job, another group does too, as demonstrated a few days before this paper was due, where the experiments for this paper clashed with those of another researcher submitting a paper for the same conference). Finally, access is sometimes regulated by having to write research proposals, which is a time cost and may not be successful.

Another possibility is a *virtual* cluster. Essentially, this is the renting of CPUs when needed from a supplier — not the physical CPUs, but time on CPUs, accessed remotely through the internet. This paper explores the use of one such technology, Amazon's Elastic Computing Cloud (EC2), as an example of how this could be used for scientific computing. In summary, the paper argues that this technology is a useful complement to dedicated individual clusters and large shared systems.

The basic approach of EC2 is that the user stores their data within the Amazon system, paying relatively low rates for data storage. When a user has a job to run, they can pay for as many computing nodes as needed, which are charged at an hourly rate. While nodes are being rented, the user has complete control of the system, having root access to the nodes. These nodes can thus be configured as the user desires with whatever packages and system software needed. In particular, the nodes are networked, so can communicate with each other. This allows the user, for example, to run a version of MPI [10] on the nodes and so run a job in parallel.

The attraction of the technology is that if the user does not run any jobs, the only cost is for data storage. When a job or jobs run, as many CPUs as useful can be deployed. This changes the mind-set of the researcher: the cost of the job is determined by the total computation time. If an algorithm parallelises effectively on n CPUs, a problem using that algorithm costs roughly as much to solve using n CPUs in one hour as using one CPU in n hours. m separate jobs cost as much to run sequentially as concurrently.

Contrast this with a dedicated cluster, where there is a trade-off to be made between getting a big cluster, in which many CPUs will be idle for most of the time and a small cluster which will be cost-effective, but which will take a long time to solve large jobs.

Structure of paper. This case study uses the criteria of performance, cost and cost-effectiveness, learning curve, and ease of use access to evaluate the EC2. The core of the paper is a quantitative comparison between virtual EC2 clusters and two "real" clusters at the CHPC and Meraka. It also contains a personal and subjective account informed by the experience of running real jobs in different environments. The rest of the paper is structured as follows. Section 2 gives an overview of the relevant Amazon Web Services. Section 3 briefly describes the application used for testing. Section 4 then presents the case study. Finally, Section 5 discusses and concludes.

2. AMAZON WEB SERVICES

This section gives a brief overview of the Amazon Web Services, focussing on those services relevant to the case study. Due to space limitations, it is a very restricted description and key features have been omitted (e.g. security). For more details see [1] and the Amazon Web Services site `http://www.amazonaws.com`. Weiss [11] gives a good overview of Cloud Computing.

Although there is extensive reference to S3 and EC2 in the scientific literature there have been relatively few published studies. The papers of Palankar [9] and Garfinkel [5] evaluate the data storage and access features of the Amazon Web Services for scientific computing. Choi *et al.* present a scientific collaborative lab system built on S3/EC2[4] focussing on workflow rather than performance issues. This paper focuses on the *computing* resources of EC2.

2.1 Amazon's Simple Storage Service

Though not the focus of the work, the cornerstone of the Amazon Web Services is Amazon's Simple Storage Service (S3). It provides users the ability to store large amounts of data reliably and with high availability. Data is read and written using protocols such as SOAP, REST and BitTorrent, and is also accessible via normal web browsers. For example, a web page describing some of the technical details related to this paper can be found at `http://s3.amazonaws.com/witsbioinf/saicsit2008.html`.

The storage model is a simple two-level hierarchy. Users may create *buckets*, and place data *objects* in the buckets. Strings are used as keys for both buckets and objects, and so they are easily incorporated in URLs. In the above example, the bucket with key *witsbioinf* contains an object with key *saicsit2008.html*). As slashes can be part of object keys, an arbitrary depth hierarchy can be simulated.

Each user account can have up to 100 buckets. An unlimited number of objects of up to 5GB each can be placed in each bucket.

Users are charged 15 US cents per Gigabyte per month. There is also a cost for transferring data to and from S3 (but not between S3 and EC2 or within EC2).

2.2 Elastic Computing Cloud

The Elastic Computing Cloud (EC2) is physically a large number of computers on which Amazon provides time to paying customers. EC2 is physically based in different locations in the United States. A significant part of the development of EC2 was done by Amazon's South African office.

EC2 is based on Xen virtualisation technology [2]. This allows one physical computer to be shared by several virtual

computers, each of which hosts different operating systems. Each virtual OS has its own root, and lives in its own separate universe. In principle, Xen virtualisation can allow any sort of operating system be hosted.

EC2 provides users virtual hosts based on Linux operating systems. A range of 32-bit and 64-bit kernels supporting the common Linux varieties such as Ubuntu and Fedora Core are available. Amazon has made available a number of *Amazon Machine Images* (AMIs) which can be hosted on their computers. Users can launch *instances* of these AMIs over the internet and interact with them. Users may take an AMI that has already been provided and add their own system and application software on the AMI, as well as remove some of the packages installed on the AMI. These new machine images can be packaged as an AMI which can be either private to the user or publicly available. These AMIs can in turn be launched or repackaged.

This research used two AMIs as bases: ami-75789d1c, an Ubuntu 8.04 AMI packaged by Eric Hammond; ami-3e836657, a Fedora Core 6 AMI packaged by Marcin Kowalski (as well as their 64-bit equivalents). These were modified these by installing newer versions of gcc and MPI, essential applications such as emacs, the scientific software used for this case study, some sample data and a generic user account. The 32-bit AMI used in this research is publicly available as `witsbioinf/wcd45-i386-public`, `ami-33b5515a`.

Users may launch EC2 instances – virtual computers. As of June 2008, there are 5 different *instances types* available, with different features. Amazon bases these instances on an idealised notion of an EC2 compute unit, as a unit of CPU performance. This is 'the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor'[1]. Thus, a user is abstracted from an underlying physical machine, and each time they use a particular EC2 type may actually use a different type of physical machine, which they may or may not be sharing with other users. Amazon therefore attempts to standardise the service available, though the use of 'equivalent' as a descriptor is probably too strong. The five different types of instances are given in Table 1. The variations are the number of (virtual) cores, the amount of RAM, whether it is a 32-bit or 64-bit architecture, how much storage is available, I/O performance, and price, which is charged on an hourly basis.

Once an instance is launched, the user is given a DNS address, which can be accessed using ssh. The user has root control and may run their own software. As an example, to solve a large job, the user may launch 30 instances which can communicate with the user and each other using MPI or other protocols.

The secondary storage associated with each instance exists only while the instance is active. The user may reboot the instance and keep the secondary storage, but once the instance is terminated[2], the secondary storage terminates too.

[1]Taken from the Amazon web page: `http://www.amazon.com/Instances-EC2-AWS/b?ie=UTF8&node=370375011`.

[2]An instance may be terminated at the user's request, or if it crashes. No SLA is given, but the anecdotal evidence is that the instances are highly reliable and may run for months or longer. In all the experimentation reported here, no instance crashed.

The Amazon Elastic Block Store (EBS) is a persistent storage medium for instances. An EBS is provided as a raw, block device, which the user may format with an appropriate file system. (The EBS was released after this paper was reviewed and a few days before the final copy-edited version of the paper was due and so has not been evaluated for either cost-effectiveness or performance. The absence of such a service was noted as a limitation, and the service needs proper evaluation).

Note that there is no charge for transferring data back and forth between S3 and EC2 nor between EC2 instances within the same availability zones.

2.3 Other services

Amazon provides other services, which while are useful for some scientific applications, were not used in this case study. These two most important are the SimpleDB and the Simple Queue Service.

The SimpleDB is a simple database service that allows users to store structured data and to perform queries on the data. This service provides many of the facilities of modern relational databases, and provides a web services interface to the system.

The Simple Queue Service provides a reliable, scalable queueing service between EC2 instances. This allows multiple computers to send messages to each other reliably through a web services interface.

3. THE WCD EST CLUSTERING SYSTEM

This paper uses one scientific application as a case study. It is an example of an application with very large computational needs, and representative of applications with high CPU needs, demanding on L2 cache. It scales reasonably well using MPI. With respect to input data size it has a linear amount of communication for a quadratic amount of work. The I/O demands are very modest. RAM needs for real data sets are typically in the 500MB to 1GB range.

Using one application as a case study is limiting and further experimentation is needed to explore the effectiveness of EC2 for other kinds of programs. However, focussing on one program allows us to explore the effectiveness in detail.

The application is a bioinformatics application called `wcd`, which clusters expressed sequence tags (ESTs). ESTs are short fragments of DNA, and their processing is an important application. As of June 2008, there were 53 million ESTs from over 1500 different species in the dbEST database [3] hosted at the NCBI (`http://www.ncbi.nlm.nih.gov/dbEST/`). Typically ESTs are between 300 and 500 characters in length. Since the clustering of ESTs is quadratic in number and size of ESTs, processing is computationally challenging.

The goal of clustering is to group those fragments that are related and overlap. These groups or clusters can be used to study the *transcriptome* – i.e. to help us understand the be-

Type	#cores	EC2 power	RAM	32/64	Storage	I/O	Price
m1.small	1	1	1.7 GB	32	160GB	M	10
m1.large	2	2	7.5 GB	64	850GB	H	40
m1.xlarge	4	2	15 GB	64	1690GB	H	80
c1.medium	2	2.5	1.7 GB	32	350GB	M	20
c1.xlarge	8	2.5	7GB	64	1690GB	H	80

Table 1: Specification of EC2 Instances. Type: the type of EC2 unit. #cores: the number of virtual cores. EC2 power: the rating of *each* virtual core. RAM: size of RAM in Gigabytes. 32/64: number of bits in the architecture. Storage: size of "hard disk" in Gigabytes. I/O: performance of I/O system — M is moderate, H is high. Price: in US cents per hour.

haviour of genes in cells. These clusters can also be given as input to *assemblers*, which put together the relatively small ESTs into longer *contigs* and so determine the underlying gene or genetic sequence. For a fuller description of ESTs and their use, see the survey of Nagaraj *et al.* [8].

The `wcd` EST clustering system implements sensitive measures of sequence similarity which allows good quality clustering. Parallelism is supported in two modes: a pthreads implementation for shared memory machines, and an MPI mode for distributed machines (such as a Linux cluster) and/or shared memory machines. A full description of `wcd` is beyond this paper – see [6, 7] for discussion of the underlying algorithms and biological efficacy.

For the experiments conducted here, we used the following data sets

- A686904 contains 686904 *Arabidopsis thaliana* ESTs, in total 285M of nucleotide data;
- A032, a subset of A686904 with approximately 32M of data;
- Public Cotton: a set of 30k cotton ESTs with 18M of data.

The data sets used in this paper are described in [7] in more detail and are available from `http://www.bioinf.wits.ac.za/~scott/wcdsupp.html`. A686904 is a relatively modest data set. For example, there are over 1 million *Triticum aestivum* ESTs (spring wheat) and over 8 million human ESTs.

The results of a run of `wcd` is a cluster table. Since this contains indices into the data file, the cluster tables are relatively small, typically 1–2% of the input file size.

4. EXPERIMENTATION

The primary goal of the experimentation was to explore how `wcd` performed computationally on the EC2 clusters. We are interested in the power of the individual EC2 instances, and more importantly how the parallel version of `wcd` scales as the number of processes increases. Sections 4.1–4.3 explore the computational performance of EC2 on `wcd`. Section 4.4 looks at network costs. Section 4.5 explores the usability of the system.

4.1 MPI performance: Experiment 1

This experiment compared a virtual EC2 cluster with two real clusters at the CHPC and Meraka. Figure 1 and Table 2 on the next page show the performance results of the iQudu cluster of the CHPC, the Meraka C4 Xeon cluster and an Amazon EC2 cluster. The time taken for different numbers of slave processes is shown as well as the efficiency of the run. If $T(n)$ is the time taken to run with n slaves, then efficiency, $E(n) = T(1)/T(n)/n$.

The specifications of the clusters are:

- C4: The relevant part of the cluster consists of 37 nodes, each with 4GB of RAM and two single core 3.66GHz Intel Xeon Irwindale processors.

  MPICH was used, scheduled using the Sun Grid Engine. Gigabit ethernet is the underlying architecture. The code was compiled using gcc 3.4.6.
- iQudu: The cluster consists of 160 nodes, each with two dual-core AMD Opteron 2218 processors and 16GB of RAM, using Infiniband interconnect. `wcd` was run in the mode of 4 MPI processes per node. MVAPICH was used, scheduled with LoadLeveler. The code was compiled with gcc 3.4.3.
- EC2 cluster: This was a cluster of `m1.large` nodes (2 virtual cores per node, each rated at 2EC2 units), 7.5GB of RAM per node. LAM-MPI 7.1.2 and gcc 4.2 versions were used.

In all cases, the memory used was well within the bounds of the machines. From previous experimentation, we know that `wcd`'s performance is sensitive to L2 cache size.

	EC2		iQudu		C4	
#	Time	$E(n)$	Time	$E(n)$	Time	$E(n)$
1	100072	1.00	76243	1.0	55680	1.0
3	34874	0.96	24656	1.0	18903	0.98
7	15217	0.94	10652	1.0	8178	0.97
15	7380	0.90	4938	1.0	4213	0.88
31	3702	0.87	2538	0.95	2260	0.79
47	2855	0.75	1870	0.87		
63	2198	0.72	1411	0.86		

Table 2: Scalability of wcd on three different clusters. Times are shown in seconds.

These results shows that the EC2 cluster acquits itself well. With 63 slaves (32 nodes, each with 2 cores), efficiency was

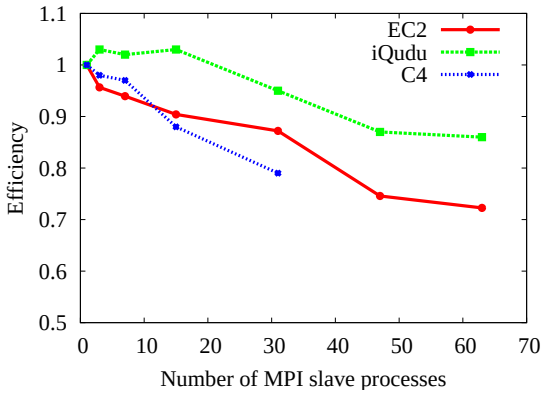


Figure 1: Comparative performance of three clusters on the A686904 data set. The x-axis is the number of slaves used, the y axis shows the efficiency of parallelisation.

72%. For 31 slaves, the efficiency on the EC2 cluster is about half way between the efficiency of iQudu and C4. iQudu is an ideal environment for `wcd`.

4.2 MPI Performance: Experiment 2 – shared memory

Some of the EC2 instances are multicore machines with shared memory. Exploring the cost/benefits of using a single computer with multiple cores compared to several computers with single cores is important. `wcd` is a cache-hungry application, and so we would expect it to suffer on a shared memory architecture compared to a distributed environment.

Figure 2 shows the results of experiments with the A032 subset of the A686094 dataset to test the performance on a multicore system. Here we used the `c1.xlarge` architecture (8 cores, 2.5 EC2 units each). To measure the performance we used MPI to run different numbers of processes on a single EC2 instance with 8 cores, and report the time taken and the efficiency (see curve AX-1IMC in the figure – 1IMC=1 instance, multiple virtual cores). For comparative purposes we ran the same experiment on three reference architectures:

1. A 2.33 GHz Intel Xeon E5345[3]. This is a dual-processor machine with four cores per processor, so 8 cores in total. This is the curve labelled E5345.

2. 8 `c1.xlarge` instances, with one process per node (even though these have 8 virtual cores). This is the curve labelled AX-MI1C (multiple instances, one virtual core each).

3. 4 `c1.medium` instances, each with 2 virtual cores. Here we ran 1 or 2 processes per instance. This is a 32-bit rather than 64-bit architecture. Note the cost of four `c1.medium` instances is the same as one `c1.xlarge` instance. This is the curve A4CM.

[3]This happens to be a core compute server at Wits Bioinformatics, but by chance it's also what /proc/cpuinfo calls the `c1.xlarge` instance, which it may or may not be.

Num threads	E5345	EC2 x.large
1	198	189
2	105	104-187
3	75	88-183
4	60	73-187
5	52	67-186
6	49	54-128
7	48	60-187
8	48	55-134

Table 3: Performance of Pthreads parallelisation – a comparison of an EC2 c1.xlarge and the Intel Xeon E5345. The table shows the time taken to process the Public Cotton EST set.

An important result is not shown in the Figure: for the E5345 and AX-MI1C, the experiments were run several times but the difference between maximum and minimum times was less than 2%. For AX-1IMC, as the number of cores increased the variability increased too – for 7 slaves over 11 different runs, the average time was 162 seconds with a standard deviation of 22 and a difference of 65 from smallest to biggest. Variability was also seen on the A4CM curve, but not as high. For the high-variability cases, the experiments were run at least 10 times and the average is shown.

4.3 Pthreads

To explore the use of virtual cores further, we also compared the performance of `wcd`'s pthreads parallelisation, running several threads at the same time. From previous experimentation, we know that `wcd`'s pthreads parallelisation is not particularly strong – depending on the architecture it works well for 2-4 cores but not beyond that. Table 3 shows the performance of the pthreads implementation on the Public Cotton data set using a single `c1.xlarge` instance and the Intel E5345 Xeon.

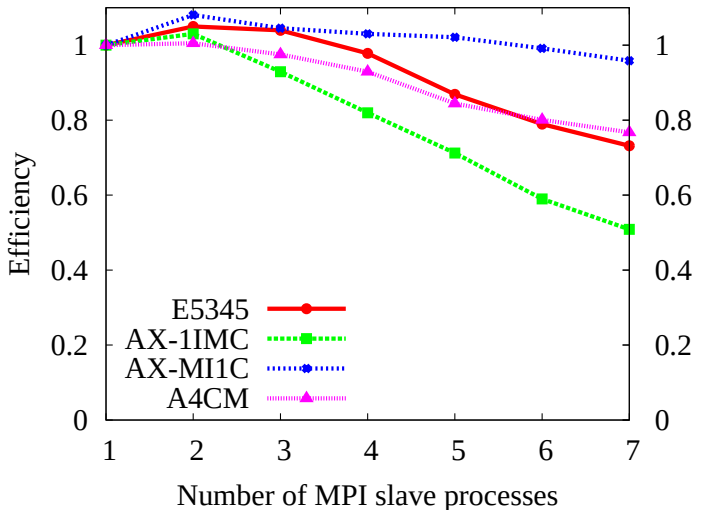
As can be seen the pthreads version scales reasonably to the 4 core mark on the E5345, with very modest improvement to the 6 core mark. For the `c1.xlarge`, scaling is not as good. Moreover, the variability was extremely high. For example, with 4 cores over 12 runs, the range was 73s-187s, with an average of 149 and standard deviation of 47! Variation was seen across different EC2 instances and on the same instance. In all cases the EC2 instance was otherwise idle (though the underlying physical machine may well have had different loads). By contrast, variability on the E5345 was small (2 seconds at most).

4.4 Network speed

As `wcd` and many scientific applications have large input data sets, bandwidth is an important issue. Although `wcd`'s output is relatively small, many applications will also produce large data sets as output. This section reports on network bandwidth results. Network speed impact on interactive use is explored in Section 4.5.

Network speed is particularly important for the South African community where currently bandwidth costs are notoriously high. It is worth mentioning that one potential advantage of using S3 an EC2 for South African researchers or researchers

(a) Efficiency of MPI on different architectures



(b) Absolute time

#s	E5345	AX-1IMC	AX-MI1C	A4CM
1	630	577	577	591
2	300	280	267	294
3	202	207	184	202
4	161	176	140	159
5	145	162	113	140
6	133	163	97	123
7	123	162	86	110

Figure 2: Evaluation of multicore performance using MPI, using the A032 data set. The figure on the left shows efficiency. The x-axis is the number of slaves, and the y axis is efficiency of parallelisation. The table on the right shows the number of slaves versus the actual time in seconds. E5345: Performance on a single Intel E5345 8-core machine using different numbers of cores. AX-1IMC: Performance on a single c1.xlarge instance using different numbers of cores. AX-MI1C: Performance on multiple c1.xlarge instances using 1 process/core per instance. A4CM: Performance using four c1.medium instances.

from other high-cost bandwidth environments is that when working with international collaborators, data can be stored on S3 and processed using EC2. This means that South African researchers can be actively involved with their collaborators without having to incur delays in networking.

In the various experiments, network speed was variable and it was not possible to isolate what delays are due to traffic shaping at Wits, what due to the size of our international pipe and what is due to network contention at Amazon. However, it seems that a good part is due to local traffic shaping. For example, download from EC2 on a home DSL line (Telkom Fastest DSL) was significantly faster line than from Wits (but not upload). A full study such as that shown in [5] is necessary to characterise network performance. The results here are representative of performance at different times of day (and night), and are therefore presented as realistic samples rather than average performance.

Wits has a 32 Mega*bits* per second international bandwidth pipe that it rents from TENET (`www.tenet.ac.za`), and there is fairly heavy local traffic shaping. At the time of testing, the Telkom ADSL line was quoted by Telkom as 4096 kilo*bits* per second download, and 256 kilobits per second upload. Testing with `www.speedtest.net` showed these are credible figures. The results below are quoted in Kilobytes per second.

- From Wits: using scp to an EC2 instance
  - 8M file: from 110KB/s to 260KB/s
- To Wits: using scp from an EC2 instance
  - 100KB/s
- With a Telkom Fastest DSL line: using scp to an EC2 instance
  - 8M file: 52KB/s-60KB/s
- With a Telkom Fastest DSL line: using scp from an EC2 instance
  - 8M file: 152KB/s-320KB/s
- From a Canadian site (`cs.ubc.ca`): to an EC2 instance
  - 450KB/s for a variety of files
- Between EC2 nodes:
  - From m1.small to m1.small: 17MB/s
  - From m1.small to c1.xlarge: 30MB/s
  - From c1.xlarge to c1.xlarge: 49MB/s
- EC2/S3 transfers – between 10MB/s and 20MB/s – approximately 14M for large files.
- Downloading large EST data sets using ftp from dbEST at NCBI `ftp.ncbi.nlm.nih.gov`:
  - From Wits 123 KB/s
  - From an EC2 instance 5MB/s (40 times faster)

Garfinkel [5] carried out a thorough evaluation of network performance, focussing on HTTP PUTs and GETs. His findings with respect to EC2-S3 communication is broadly in line with what is reported here. Data rates of reads from

S3 ranged from 212KB/s for a site in the Netherlands to 412KB/s from Harvard, to 651KB/s from MIT. Write rates ranged from 383KB for the Dutch site to 620KB for Harvard to 2200KB/s for MIT.

Garfinkel believed that the limits on bandwidth was not caused by limitations of bandwidth at the host sites – the implication being that limits were either at Amazon or due to problems in peering arrangements. However it is clear from the results reported here that as of mid-2008 international bandwidth from South African universities is a limitation. (I repeated the transfers from another South African university at which I have an account and found the Wits transfer times were significantly better. From the figures available at the `tenet.ac.za` site, Wits seems to have good internet connectivity by South African standards.)

The following measurements for the transfer of very large files put these transfer rates into context:

- scp transfer from Wits to Meraka: ∼50KB/s
- scp transfer from Wits to CHPC: ∼45KB/s
- scp transfer rates between different subnets at Wits: 11MB/s
- scp transfer rates between different computers on my own subnet: 47MB/s

Currently, transfer rates to S3/EC2 are competitive with transfer rates to central South African facilities such as CHPC and Meraka. This can be expected to change shortly: Wits is already on the new South African Research Network (SANREN) and we are able to transfer data from `mirror.ac.za` at a rate exceed 5MB/s, and most South African research institutions should be on SANREN by the end of 2008. International bandwidth for academic institutions is expected to improve significantly in 2009 with the launch of SEACOM.

Transfer rates between EC2 instances are competitive with transfer rates between computers on my own subnet at Wits, but are more variable.

4.5 Usability

The usability of a system is crucial, even in applications where users can be expected to have high levels of expertise. EC2 is not primarily designed for scientific applications, which makes an evaluation important. The results here are anecdotal as they report my subjective experiences in learning to use S3 and EC2, and then carrying out the experiments, so is written as a personal account.

4.5.1 Learning to use S3 and EC2

In the last 18 months I had learned to use the two different job schedulers at CHPC and Meraka and so can compare the experiences. Undoubtedly, learning to use S3 and EC2 to the level of making your images is more difficult than learning to use a job scheduler.

In comparison, it was much more difficult to learn to use MPI in the first place, so anyone who has the capacity to learn MPI can learn to use EC2 without problem. The documentation is good and there are user forums.

4.5.2 Remote access

Using EC2, Meraka or CHPC all require remote access. The problem of data transfer was discussed in Section 4.4, but network latencies also affect interaction with remote computers — long delays making remote working very painful. I had no problems with remote interaction, for example issuing shell commands and editing files using emacs. It was not as fast as working on my own machine, but it was not an unpleasant experience. It certainly was not worse than using the Meraka C4 or CHPC clusters.

One benefit of the fact that the computers are based in North America is that access from those computers to remote resources is much easier. For example, using Fedora and Ubuntu utilities like *yum* and *apt-get* to install new packages is much faster than what I am used to. The bandwidth reported earlier detailing costs of downloading large data sets from NCBI is worth emphasising here: taking 1 minute rather than 40 minutes to download large files makes productivity much higher and scientific work more fun.

4.5.3 Ease of use

Once the system is configured, EC2 is on the whole much easier to use than remote shared clusters. The instances do not have to be shared and the machines can be configured as desired as the user has root permissions.

Debugging and developing locally is clearly the best due to short latencies, and the availability of GUI tools. I found developing code and then testing using a job scheduler on a remote, shared cluster particularly difficult since you are separated from the running program. The type of information you can get from the system is less and the debug-run cycle is longer. EC2 is definitely superior to remote, shared clusters for testing purposes.

An advantage of EC2 from an experimenter's view is that you can run as many instances as needed, which requires a psychological shift. If you are running several experiments and repeating them, rather than running them sequentially, you can launch additional instances and run them at the same time. And it costs roughly the same to run n jobs one after another than to run n jobs at the same time. This makes experimenting much easier and saves time.

Amazon provides a range of command-line utilities for bundling and controlling EC2 images and instances, and has sample scripts for communicating with S3. There are a number of tools and software packages that have been developed by third party developers to aid the use of Amazon Web Services. Some are free and others require payment. Two which I found useful are two Firefox add-ons that allow control of EC2 and S3 using a GUI rather than the command line. Elastic Fox (see Figure 3) allows a user to control AMIs, both public and private. A user can launch new instances, and terminate and monitor existing ones. S3Fox (see Figure 4) provides a simple GUI which allows users to transfer data between their desktop and S3, to create buckets, delete files and set permissions.

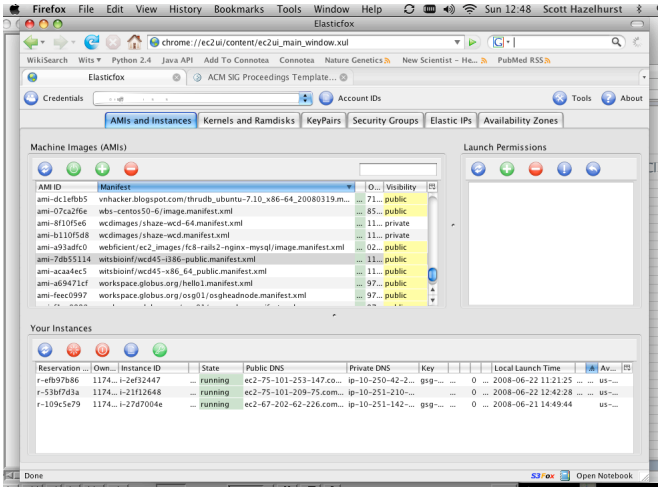


Figure 3: Screenshot of Elastic Fox: Elastic Fox provides a GUI which allows uses to register, deregister and launch new instances.

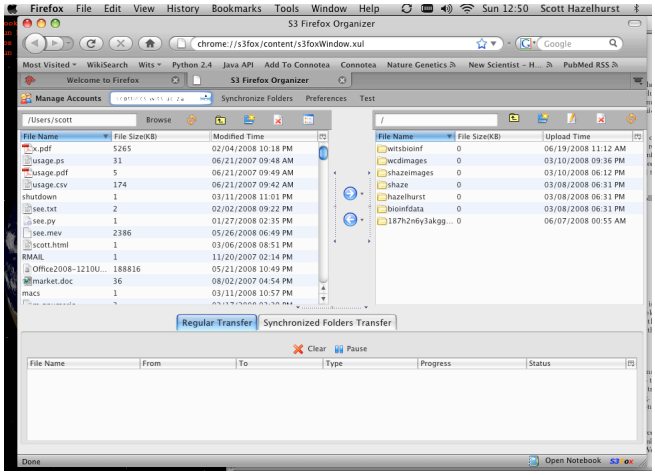


Figure 4: Screenshot of S3 Fox: This shows my local files in the window on the left and the files on S3 in the window on the right. The user can move files between places, change permissions etc.

Amazon provides good usage reports so users can track their usage and monitor their costs.

Although I found using EC2 easier than using a shared cluster with a job scheduler there are some disadvantages. The biggest problem experienced in the experiments was the fact that the "hard disks" associated with EC2 instances did not have an existence beyond the instance's life-time. For user data, this meant copying back and forth between S3 and EC2. This was easy to do in a script and there are open source tools like `jets3t` that help this. For system software, it is more difficult since changes may mean rebundling and registering the new AMI which I found took between 4 and 15 minutes depending on the size of the image and the power of the EC2 instance. It is possible to put some of the system hierarchy on S3 (e.g. the `/usr/local` or `/opt` hierarchy) and then put a script called from `rc.local` which will automatically fetch and install the software from S3. This takes a little work to get right once, but the fast interconnect between S3 and EC2 means that provided the software is not too big, it is efficient and free to do. The Elastic Block Service released by Amazon after experimentation was done may address this problem.

Some of the other issues I found were:

- System administration becomes the user's responsibility. Both CHPC and Meraka's clusters have competent and responsive system administrators. So, the benefit of being able to do your own system administration when you want is substantial, but it comes at a real cost as it may mean undesired work.
- Shared clusters come not only with hardware but with commercial software that might otherwise not be affordable, such as specialised compilers and libraries. This is not a major factor for me, but others might experience this.
- A job scheduler is useful in some circumstances (of course, job schedulers such as Oscar can be implemented on EC2 AMIs). Perhaps the most serious incident I experienced was wanting to launch a job on 32 EC2 machines late on a Friday afternoon that I expected to take 3 hours to complete. On a shared cluster there would be no problem — I could come in on Monday morning to see my results. However, paying US$24 an hour (32×0.8) would lead to a big bill on Monday morning. This meant I either had to kill my job, or make sure I could check my status later that night and terminate the instances.
- I had a little difficulty in using tools like S3Fox and Amazon's sample Python scripts for transferring data to and from S3 from behind the Wits firewall.

4.6 Cost

The cost-effectiveness of the EC2 and S3 depends on overall usage patterns, and there is a trade-off between cost and compute power.

Comparison with using shared clusters. For scientists there is commonly no direct cost for the use of large shared clusters such as C4 and iQudu. However, there may be indirect costs. Access is often restricted and dependant on competitive research proposals. The time taken to write proposals must be taken into account, and there is no guarantee of access at all.

The relative performance of a high-performance shared cluster and EC2 depends on the nature of the application. For very highly scalable applications (e.g over 60 processes), systems like iQudu will provide better performance than EC2. However, EC2 appears to perform better than commodity systems. Furthermore EC2 has the advantage that it is much more extensible. As an example if we have an application that has moderate parallelisation (e.g. 16 processes) which has to be run several times on many different data sets, then

far better performance can be expected from EC2 than most high-performance centres.

Comparison with dedicated clusters. A direct comparison is difficult because for most researchers costs such as space, power and insurance are free or subsidised by the host institution. It is also difficult to quantify costs such as air-conditioning and UPS which should be taken into account. The calculations below exclude these extra costs and so favour the use of dedicated clusters.

As a comparison, the E5345 used in the experimentation above (an Intel E5345 2.33GHz Xeon – dual processor, 4 cores per processor – with 4GB of RAM) cost me roughly US$2700. The results above show that for applications with 1-2 processes at a time, its performance is roughly within 10% of that of the `c1.medium` instance which costs US$0.20 per instance-hour. US$2700 is about 13500 instance hours, just over 1.5 instance-years. Thus if a server machine like this has a life-time of 3 years, the break-even mark for a machine is about 10-50% utilisation, depending on number of cores used. If a machine is used more, it is more cost-effective to have your own; if less, rent from Amazon.

This will obviously vary from application to application. In these experiments, for example, the `c1.xlarge` did not perform as well as the E5345 when multiple processes were run on one instance. In this type of application, with a lower utilisation it would be more cost-effective to have a dedicated machine.

Practical issues. Payment for the use of Amazon Web Services is through a credit card. Practically this means using a personal credit card and getting a refund from a host institution. This was not been a problem in the research. But how could postgraduate students use the system? Given that it is easy to spend US$1000 of compute time in a weekend, questions of financial control must be addressed.

Bill for this research. The total bill for the work done on S3 and EC2 for this research was approximately US$180, about 98% of which was for the use of EC2 compute resources. This includes the costs of learning the system (fairly small) and running many experiments on large data sets, on different instances and replicating for accuracy.

The cost of clustering the A686904 data set is approximately US$10-$13 (less than R100 rand).

5. CONCLUSION

The launch of Amazon's Web Services, in particular the Simple Storage Service and Elastic Computing Cloud provides the scientific community with another possible platform for high-performance computing needs.

The work of Palankar *et al.* [9] shows the strength and shortcomings of the technology as a means of storing large amounts of data reliably on S3. Once on S3 the data can be processed by different EC2 instances.

This paper has shown the viability of EC2 for highly scalable applications, with good speed-up achieved for up to 63 processors. This scalability compared well with existing shared facilities.

The advantage and disadvantage of the model is that it makes the price of computing explicit. It makes entry and scaling-up fairly easy and cheap. It does away with the need for backups, UPS, air-conditioning and power. You pay for what you get when you use it. If you do not need compute time, you do not need to pay.

The advantage over shared resources is researchers are guaranteed access time when they needed; there is no competition for access and nor a gate-keeping process for access. Users have root access to the machines. Shared resources such as CHPC iQudu and Meraka C4 are, however, cheaper – no direct costs – once you do have access and come with system administration and other support. Of course, from the funder of the shared facility, cost-effectiveness compared to EC2 will depend on the load of the cluster.

An important criterion to emphasise is how much human time it takes to solve a particular problem. In my experience – from a relatively resource-poor environment – it is worth paying a premium to complete various experiments in shorter elapsed time. Time is probably more a limit than money for me. Experiments that drag on over weeks are more likely to be interrupted by other tasks like teaching and administration and also make it harder to remain competitive in research. For me EC2 offers a big benefit for this type of situation.

Compared to smaller, dedicated clusters, the main disadvantages are communication delays in transferring data and lack of GUI tools. Whether EC2 is cost-effective compared to a dedicated cluster depends on the utilisation levels of the machines in the cluster. Where clusters are idle for the majority of the time, the EC2 is an attractive solution.

There are some disadvantages besides the communication costs. The performance of multiple virtual cores on an instance appeared very variable for the single application tested in this research. The lack of persistent storage for instances, except through backup to S3, was noted as a drawback. The recently released Elastic Block Service addresses this need but was not evaluated as it was released a few days before the camera ready version of this paper was due.

One practical problem with using EC2 for extensive research will be funding. Funders understand the need to provide money for powerful computers to research groups; whether they will feel comfortable with providing similar magnitude of funds to pay Amazon bills of PIs credit cards remains to be seen.

There is still the opportunity – indeed need – for sharing. For example, for bioinformatics research, EC2 offers the potential for doing many searches in parallel. However, this requires large amounts of data to be stored in S3. The cost of storing the major bioinformatics databases that Wits Bioinformatics mirrors would approximately be R20000 per annum. Ideally this is something that could be shared between many users.

My view is that services such as EC2 and S3 will not replace

either dedicated clusters, or large shared super-computing facilities. Local, dedicated clusters are needed for small-scale experimenting, for learning and teaching. The absolute costs of small clusters are often small enough that even if EC2 is theoretically cheaper, the benefits of having your own cluster outweigh this (though recent power crises locally have moved the balance in the EC2 direction). Large, shared facilities can support high utilisation levels and are probably an effective way of funding researchers. They can also meet applications which have very high I/O demands and low-cost communication needs.

However, these type of web services complement both types of clusters. For many scientific research applications, EC2/S3 will provide a much more cost-effective path that leads to solutions in faster time.

6. ACKNOWLEDGEMENTS

This research was supported by grants from the National Bioinformatics Network, the National Research Foundation (GUN2069114), and the Wits University Research Committee. The Centre for High Performance Computing and Meraka provided generous access to their equipment. Anonymous referees made useful suggestions. Thanks to James Greenfield, Marcin Kowalksi and Greg Kempe for help and useful comments on the paper (but the the views and opinions expressed, and any errors are solely those of the the author).

7. REFERENCES

[1] Amazon. Amazon Elastic Compute Cloud developer guide. `http://s3.amazonaws.com/awsdocs/EC2/2008-02-01/ec2-dg-2008-02-01.pdf`, Feb. 2008. API version 2008-02-01.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[3] M. Boguski, T. Lowe, and C. Tolstoshev. dbEST–database for expressed sequence tags. *Nature Genetics*, 4(4):332–3, Aug. 1993.

[4] J. Y. Choi, Y. Yang, S. Kim, and D. Gannon. V-lab-protein: Virtual collaborative lab for protein sequence analysis. In *Proceedings of the IEEE Workshop on High-Throughput Data Analysis for Proteomics and Genomics*, Nov. 2007.

[5] S. Garfinkel. An evaluation of Amazon's grid computing services: EC2, S3 and SQS. Technical report, Harvard University, 2008. Technical Report TR-08-07.

[6] S. Hazelhurst. Algorithms for clustering EST sequences: the wcd tool. *South African Computer Journal*, 40:51–62, June 2008.

[7] S. Hazelhurst, W. Hide, Z. Lipták, R. Nogueira, and R. Starfield. An overview of the wcd EST clustering tool. *Bioinformatics*, 24(13):1542–1546, July 2008. doi:10.1093/bioinformatics/btn203.

[8] S. Nagaraj, R. Gasser, and S. Ranganathan. A hitchhiker's guide to expressed sequence tag (EST) analysis. *Briefings in Bioinformatics*, 8(1):6–21, 2007.

[9] M. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for science grids: a viable solution? In *Proceedings of the Data-Aware Distributed Computing Workshop (DADC)*, Boston, June 2008.

[10] M. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2003.

[11] A. Weiss. Computing in the clouds. *netWorker*, 11(4):16–25, 2007.