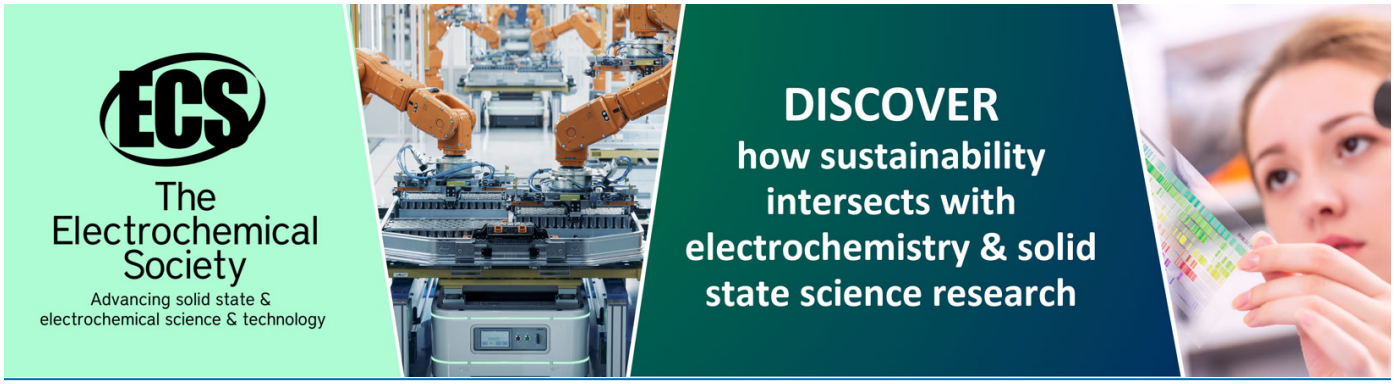
# ATLAS@AWS





Journal of Physics: Conference Series **219** (2010) 052020 doi:10.1088/1742-6596/219/5/052020

# ATLAS@AWS

**Jan-Philip Gehrcke**[1], **Stefan Kluth**[1,2], **Stefan Stonjek**[2]

[1]Universität Würzburg, 97070 Würzburg, Germany
[2]Max-Planck-Institut für Physik, Föhringer Ring 6, 80805 Munich, Germany

E-mail: gehrcke@mpp.mpg.de, skluth@mpp.mpg.de, stonjek@mpp.mpg.de

**Abstract.**

## 1. Introduction

Cloud computing was an intensely discussed topic at this conference. This was inspired to a large degree by Amazon Web Services (AWS) [1], which went for its core components from the beta testing phase into commercial production including the availability of service level agreements (SLAs). AWS offers several computing infrastructure services via well documented web interfaces, i.e. these services are available through customized http requests and their corresponding server reply messages. All services are charged for according to actual usage, e.g. per GB·time for storage or per CPU·time for computing (there are more cost relevant factors). Using AWS is described in detail in [2]. The services relevant for scientific computing offered by AWS are these:

**S3** The Simple Storage Service S3 stores data as "objects" in "buckets", which are created by the user. There is no hierarchy of buckets. Data can be uploaded, downloaded or deleted via corresponding RESTful http requests [3]. Access to the data can be controlled with access control lists (ACLs) or by creating so-called signed URIs, which give access to selected data for a limited period of time. The service is charged in the US with \$0.15 per GB·month, and \$0.10 (\$0.17) per GB of transfer into (out of) S3. Transfer to/from EC2 (see below) within the same region is not charged for.
The performance of internal data transfer between S3 and EC2 has been evaluated as up to 12 MB/s for single streams and reaching a maximum of about 50 MB/s per EC2 instance [4, 5].

**EC2** The Elastic Compute Cloud EC2 is a service where registered users can launch virtual machines (VMs) from custom machine images (Amazon machine image AMI). Interaction with the service is done with http queries which give full control over the running VMs referred to as instances. The user can get root access to the VM and has to configure network settings. The VM is run in a Xen virtualisation environment with a kernel provided by AWS. The AMIs can be obtained from AWS prebuild from several popular Linux distributions or they can be build using AWS software from a running system. An AMI provides one or more EC2 compute units (ECUs) corresponding to a $1.0 - 1.2$ GHz 2007 Opteron or Xeon processor, $1.7 - 1.9$ GB RAM per ECU and $160 - 211$ GB of disk storage per ECU. Pricing in the US is \$0.10 per ECU·hour for standard instances. By booking instances in advance

for up to three years the cost can be halved. The data stored on the instance disk is lost when an instance is stopped (shutdown). In order to keep data over shutdowns they have to be saved in S3 or on an so-called elastic block store (EBS) device, which behaves essentially like an additional disk.

**SQS** The Simple Queue Service offers a reliable messenging service based on down- and uploading of small data objects (up to 8 kB) via http queries. This system will not be used in our work.

**SimpleDB** The Simple Database (SimpleDB) provides a database with http query access. The SimpleDB does not offer the functionality of a complete relational database with SQL interface.

Running the ATLAS software on AWS could be an interesting alternative to the current model of the Worldwide LHC Computing Grid (WLCG) [6]. In the WLCG many ($\mathcal{O}(100)$) sites run one of three grid middleware stacks (glite [7], Nordugrid [8] or OSG [9]) and offer computing and storage services to the LHC experiments. The grid middleware development projects are currently nearing completion and a transition towards stable production systems is expected. Experience with the existing middleware and the experiments use of the grid shows that the WLCG can support the expected computing needs of early LHC running. However, the operation of the many middleware installations and the corresponding experiments applications needs a large amount of highly qualified personnel due to the complexity of the software. As a rough estimate two full time equivalents (FTEs) are needed to support a grid middleware installation and a few virtual organisations (VOs) at any given site. These topics have been discussed extensively at this conference.

In contrast AWS cloud computing offers a simple and well documented http based interface to computing and storage services. As will be shown below this simplicity of interfaces allows to construct correspondingly simple applications to run experiment software. It is clear that cloud computing has the potential to provide computing and storage for the LHC experiments with significantly fewer FTEs to operate the system.

The use of virtualisation to allow cloud computing users to run their own OS variants is an essential element. Sites, e.g. AWS, are free to standartise on a particular OS without forcing users to port their applications. The only requirement is that the users OS is compatible with the virtualisation environment. By providing an isolated environment virtualisation makes it possible to give root privileges to users, because the isolation layer prevents intrusion.

At this time there are only a few cloud computing services available. AWS is well known for its functionality, reliability and ease of use. Comparable alternatives from academic computing centres have appeared only very recently. We have thus chosen to evaluate cloud computing using AWS. A full writeup is available [10].

## 2. System Design

The main subject of this study was to install and run a release of the ATLAS offline software on AWS, and to use the EC2, S3 and other services to organise a small production of simulated events from a client machine at our institute. The system design follows the usual client-server structure, where servers running on AWS are controlled by clients running outside of AWS.

The server AMI based on SLC4 will be built using AWS client tools, see below. Since the total size of an AMI is limited to 10 GB and an installed ATLAS software release occupies 7 to 8 GB we decide to work with an universal SLC4 based AMI and install an ATLAS software release into the running instance. Also, the image creation process using a slightly outdated version of Linux (SLC 4.6 or 4.7) is somewhat tedious and in this way we avoid repeating this step for every release.

The client software should support the basic functions of a production system, i.e. submit jobs with individual configurations, monitor their progress, upload of input data sets and download of output data sets and logfiles. The functions are offered to a user as a commandline interface (CLI). Additional monitoring and interaction with EC2 and S3 can be done with plugins for the firefox web browser (Elasticfox and s3fox), which can be obtained from the AWS web pages.

Several libraries for popular programming languages exist, which implement the documented APIs for EC2, S3 and the other services. We choose to use the library boto [11] for use with python. The boto library implements the construction of the http requests to AWS to perform operations such as starting and stopping AMIs, up- or download of data sets from/to S3, or using EBS or SimpleDB functions.

## 3. Image Preparation

For the preparation of an AMI one can start from one of many AMIs offered directly by AWS or made available by AWS users. As an alternative an AMI can be built from an existing system. The ATLAS offline software is developed, built and tested on the SLC4 [12] Linux OS available from CERN. It turned out to be difficult to successfully install and run the ATLAS offline software (release series 14) on SL5 or Fedora 7, 8 and 9. We thus chose to build an AMI from SLC4.

We installed a fresh SLC4 system on a virtual machine [1] using the SLC4 installer and http access to the online repository of packages. Security options such as firewalls or SELinux are not used. In addition to the base system a complete set of gcc packages is installed.

In order to generate an AMI from the running system the AWS AMI tools package has to be installed. This in turn required updates to the installations of tar and ruby. Both where replaced by recent versions (tar 1.20 and ruby 1.8.7) compiled directly from source on the running system.

The resulting SLC4 AMI had problems with automatic hardware detection by the kudzu service after starting on AWS for the first time and as a result there was no network connection. AWS provides different virtual hardware compared with VMware player. SLC4 detects these changes but fails to automatically configure the new hardware. After disabling the kudzu service the SLC4 AMI could connect to the network on AWS.

In order to prepare the AMI for execution of ATLAS jobs we installed the AWS API tools, a recent version of python (2.5.2) and the boto python library. Then our own shell and python scripts (server software) are installed, which are triggered after system start via a modified rc.local script.

The AMI is created from inside the running system using the AWS AMI tool command `ec2-bundle-vol`. Due to an interference between the SLC4 kernel with high resolution timer and the VMware virtualisation the SLC4 clock runs too slow. This can cause the upload of the resulting AMI to AWS S3 to fail. As a workaround the clock on the system in the VMware player is frequently reset to correct values. Since the SLC4 kernel is not run on AWS this problem does not appear for AMIs launched on EC2.

As a last step the ATLAS offline software has to be made available to instances of our AMI. An instance is launched and then used interactively to import an ATLAS software release on an EBS with the ATLAS software distribution tool pacman [13]. The contents of an EBS can be easily archived to S3 as a so-called EBS snapshot and restored from there to another EBS. In this way every instance can be connected with an EBS with an ATLAS software release as long as this release was archived in S3 before.

[1] VMware player

## 4. Server Software

The server software consists of a series of shell and python scripts, which are initialised after system start via a modified rc.local script. The first script awsac-autorun sets up environment variables and reads the so-called user-data string, which can be passed to launching AMIs via the EC2 API. This string contains the location of an archive which was prepared on the client. This archive is downloaded to the server and contains job processing scripts as well as the job scripts themselves. The script awsac-all-instances-autorun obtains information about the running instance and starts the script awsac-processjobs, which connects the required ATLAS software releases as EBS devices and starts the actual ATLAS jobs. Awsac-processjobs also monitors the jobs and puts job status information into SimpleDB. Once a job has finished its output files and logfile are transferred to S3. When all jobs on an instance are finished awsac-processjobs unmounts the EBS device, uploads its own logfile to S3 and initiates a shutdown of the instance.

## 5. Client Software

The client software is written in python using the boto library. Its purpose is to help the user to prepare input data for a run on AWS, to launch instances on AWS, to upload input data sets and to download job output. In addition the client tools can display job monitoring information.

In order to start a production on AWS the user has to prepare job shell scripts, which configure and run the ATLAS offline software and finally collect output files in an archive, e.g. results.tar.bz2. A small file called job.cfg connects releases of the ATLAS offline software as EBS snapshots with the job scripts. These files are bundled together with awsac-all-instances-autorun and awsac-processjobs into an archive sessionarchive.tar.bz2 for upload to the instances. The file session.ini stores information about the instance type, AMI, S3 bucket, EC2 user and total number of jobs.

Once these files have been created invocation of `awsac-session.py --start -i session.ini -a sessionarchive.tar.bz2` will make sessionarchive.tar.bz2 available on S3, check the availability of the requested AMI and then launch the required number of instances. These will obtain the sessionarchive.tar.bz2 as described above and start the jobs. After completion the command leaves a file behind which stores all information needed to interact with the running instances.

Monitoring of running jobs can be done by issuing on the client e.g. `awsac-session.py --check -c session-081215_1646--eventgen--1816.cfg`. This will query entries in SimpleDB which have been stored by the server software as the jobs were progressing.

When the jobs have completed successfully the results can be extracted from S3 to the client with e.g. `awsac-session.py --getresults -c session-081215_1646--eventgen--1816.cfg -o sessionresults`. The job output archive (e.g. results.tar.bz2) and the corresponding logfiles will appear on the client.

Finally, when the results of the run have been verified the ressources on AWS can be cleaned up completely using e.g. `awsac-session.py --cleanup -c session-081215_1646--eventgen--1816.cfg`. This avoids charges for storing data in S3 or SimpleDB.

## 6. Experience

The project was realised by a summer student and two physicists with some background in HEP computing. As members of the ATLAS collaboration and managers of a local Tier-2 site we knew the ATLAS offline software and ATLAS usage of grid computing in the WLCG. All of us had no prior experience with cloud computing and the AWS services.

The simple and well documented AWS interfaces for EC2, S3 and SimpleDB allowed us to agree quickly on the basic system design. The availability of the boto python library to

encapsulate the AWS APIs behind simple python methods was essential to set up a working prototype.

We encountered greater difficulties in the creation of the SLC4 based AMI, because this comparatively old version of Linux is not directly compatible with the AWS API and AMI tools. In addition, the hardware detection and automatic configuration algorithm of SLC4 did not work on the virtual hardware provided by AWS. We did not try the cernvm [14] images, since these where not available in stable versions during project development and because they implement a special software distribution mechanism.

Another problem was the provision of the large ATLAS software releases to instances of our SLC4 AMIs. We solved the problem using the AWS EBS devices, which became available during project development. This solution still involves the transfer of the complete ATLAS software release from S3 to an EBS for every instance. In a more permanent setup on AWS one could implement virtual fileservers to provide the releases. Another alternative would be to use extensions such as s3fs [15] for S3 to emulate a filesystem based on data in S3. This would minimise the volume of transferred data needed to provide an ATLAS software release in a similar way to the cernvm solution.

As a test of our system we executed a small run with a predefined so-called job transform implementing the simulation of artificial single pion events in the ATLAS detector. The job transform performed all steps from the generation of pion 4-vectors, calculation of the detector response, running of the offline reconstruction to the final data reduction. The jobs ran successfully.

## 7. Cost Considerations

The total costs charged by AWS for the period of project development including test runs were about $35.

At the time of writing the AWS costs for computing were 10$ct/h for one ECU, see above. We take an exchange rate of 0.7€ for 1$. An AWS ECU corresponds to about 1.8 kSI2k, thus we find 4€ct/(kSI2k·h). With advance reservation over three years and running the instance for 75% of the reserved time we find 2€ct/(kSI2k·h).

On the other hand, a typical compute server purchased in 2008 in Germany and operated at a german computer center for three years will cost about 1.3€ct/(kSI2k·h) at an assumed usage efficiency of 75% [2]. This cost estimate includes a hosting fee of 16€ct/kWh and assumes that 10% of the cost of the computing servers are neccessary for infrastructure such as racks, LAN, etc. The hosting fee covers essentially only the costs for electric power and cooling and does not include administration costs.

The ratio of costs for providing compute power for three years at a usage efficiency of 75% between AWS and a purchased system operated in a computing centre is thus about 3 without advance reservation and about 1.5 with advance reservation.

The cost for storing data is 15$ct/GB/month for the first 50 TB and decreases slowly for larger amounts of data to 12$ct/GB/month for data volumes above 500 TB. AWS clearly is prepared to store data volumes in the PB range. Thus storing 1 TB for three years will cost 3780€. The data transfer into S3 costs 10$ct/GB and thus importing 1 TB will cost 100$.

A typical high performance system with direct attached storage to provide fileserving e.g. for dCache will cost 1211€/TB of net storage including infrastructure and operation costs over three years [3]. However, the usage efficiency of the storage system needs to be taken into account and we assume that on average 50% of the available storage will be used over three years. The

[2] IBM BladeCenter with HS21 servers equipped with dual Intel Xeon E5440, 16 GB RAM and a 146 GB SAS disk.

[3] Intel SSR212MC2 (MacKay Creek) with dual Intel Xeon 5405, 8 GB RAM, 12 750 GB SATA II disks connected via SAS expansion with a Xyratex DAS 1220 with 16 750 GB SATA II disks.

cost of importing the data is difficult to account for, since bandwidth costs are typically not charged to users by the computing centres.

The nominal cost ratio between AWS S3 and a fileserver operated in a computer centre for storing 1 TB for three years thus becomes 1.6. This ratio does not consider that in a more centralised cloud computing model there is less need for multiple copies of data sets in different sites. The ATLAS collaboration foresees in its computing model [16] 10 Tier-1 and about 30 Tier-2 sites. These provide among other things storage for reduced data sets for analysis, i.e. AODs, and on average one Tier-1 site and three Tier-2 sites together will each hold a complete copy of the AOD sets. This implies a factor of about twenty for the total cost of providing storage for AOD sets for ATLAS. AWS S3 is designed to scale with load and thus there would be less need to hold multiple copies of popular data sets in S3. We therefore conclude that the total cost of providing storage services for data sets with multiple copies on the grid, e.g. AOD sets, is comparable for AWS S3 and fileservers operated in a computing centre.

## 8. Summary and Outlook

As a small team of three physicists with no prior experience in cloud computing and some background in traditional HEP computing and grid computing we were able to setup a prototype job handling system for ATLAS on AWS. This was largely possible, because AWS has simple interfaces, it is well documented, and libraries implementing the AWS APIs exist. The fact that on AWS one has few constraints for choosing a version of the Linux OS allowed us to use the recommended SLC4 and thus to avoid porting problems.

Performance of the combined EC2/S3 system seems adequate even for the demanding task of ATLAS AOD processing, where high bandwidth between storage and compute nodes is needed. At the current AOD size of 0.17 MB/event a single process can read events with a rate of up to 70 Hz until the bandwidth limit of 12 MB/s is reached. Thus, assuming a bandwidth of 10 MB/s per stream, 100 processes can read 100 TB of AOD in about $10^5$ s.

Open questions are the verification of the performance scaling of many jobs accessing the same data sets, and a more in-depth study of the costs of using AWS. Furthermore, scientific computing centres may be able to set up their own cloud computing infrastructures using toolkits such as Nimbus [17] or Eucalyptus [18]. In this scenario the total cost of providing computing and storage for the LHC experiments is expected to fall below the total cost of running the WLCG, because in a more centralised system multiple data copies can be reduced and because fewer FTEs are needed for VO site support and maintainance of middleware installations.

## References

[1] aws.amazon.com
[2] *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*, J. Murty, O'Reilly (2008)
[3] en.wikipedia.org/wiki/REST
[4] blog.rightscale.com/2007/10/28/network-performance-within-amazon-ec2-and-to-amazon-s3
[5] www.scribd.com/doc/14098529/HostedFTP-Amazon-AWS-S3-Performance-Report-March
[6] lcg.web.cern.ch
[7] glite.web.cern.ch
[8] M.Ellert et al., Future Generation Computer Systems 23 (2007) 219-240
[9] www.opensciencegrid.org
[10] gehrcke.de/awsac
[11] code.google.com/p/boto
[12] linux.web.cern.ch/linux/scientific4
[13] atlas.bu.edu/∼youssef/pacman
[14] cernvm.cern.ch
[15] fedorahosted.org/s3fs
[16] CERN-LHCC-2004-037-G-085, cdsweb.cern.ch/record/811058
[17] workspace.globus.org
[18] open.eucalyptus.com