

**REPORT ON THE COLLABORATIVE RESEARCH:  
DATA INTENSIVE SCIENTIFIC WORKFLOWS ON A  
FEDERATED CLOUD**

**Submitted to**

**Korea Institute of Science and Technology  
Information**

**245 Daehangno, Yuseon, Daejeon, 305-806,**

**Republic of Korea**

**October 31, 2015**

**Gabriele Garzoglio**

**Scientific Computing Division**

**Fermi National Accelerator Laboratory, USA**



Fermi National Accelerator Laboratory  
Computing Division P.O. Box 500  
Batavia, Illinois 60510

From:  
Gabriele Garzoglio  
Scientific Computing Division  
Fermi National Accelerator Laboratory  
Batavia, IL, USA

Oct 30, 2015

To:  
Korea Institute of Science and Technology Information  
245 Daehangno, Yuseong, Daejeon, 305-806  
Republic of Korea

Dear Sir or Madam,

It is with great pleasure that I present to you this report of the Collaborative Research And Development Agreement with KISTI for 2015 (CRADA FRA 2015-0001/ KISTI-C15005). The final report, the documentation produced, the papers, and the source code developed are attached.

The collaboration focused on a multi-year program of research and development for a federated Cloud computing infrastructure. In the first year, we have demonstrated proof-of-principle integrations of Cloud and Grid systems to run scientific workflows on multiple dynamically allocated resources. In this second year, we have extended the scale and scope of the work, focusing on deployment of on-demand services in support of scientific computation and demonstrating scalability to 1,000 Virtual Machines on the Fermilab private Cloud (FermiCloud) and Amazon Web Services. In this third year, we have focused on the support of data-intensive scientific workflows and started the integration of the deliverables of all years with the Fermilab HEP Cloud Facility, a common interface to local resources, Grids, Clouds, and High-Performance Computers.

We look forward to further collaborative initiatives with KISTI and renew our interest in continuing the joint multi-year program started with this project.

Sincerely,

Gabriele Garzoglio, Ph.D. (Principal Investigator)  
Scientific Data Processing Solutions Department, Head  
Scientific Computing Division  
Fermi National Accelerator Laboratory

**REPORT ON THE COLLABORATIVE RESEARCH:  
DATA INTENSIVE SCIENTIFIC WORKFLOWS ON A FEDERATED  
CLOUD**

**CRADA FRA 2015-0001 / KISTI-C15005**

**STEVEN TIMM, GABRIELE GARZOGLIO,  
FERMI NATIONAL ACCELERATOR LABORATORY**

**SEO-YOUNG NOH, HAENG-JIN JANG,  
KISTI**

Table of Contents

1. Executive Summary .....	2
2. Introduction .....	2
3. Technical deliverables .....	2
3.1. Data-Intensive Scientific Workflows on Federated Clouds .....	3
3.2. Interoperability and Federation of Cloud Resources .....	5
3.3. On-demand Services for Scientific Workflows .....	6
4. Qualitative and quantitative output .....	7
5. Budget Allocation .....	9
6. References .....	9

## 1. EXECUTIVE SUMMARY

The Fermilab Scientific Computing Division and the KISTI Global Science experimental Data hub Center are working on a multi-year Collaborative Research And Development Agreement. With the knowledge developed in the past years on how to provision and manage a federation of virtual machines through Cloud management systems, this year they have started to build a large-scale production infrastructure to handle scientific workflows of stakeholders to run on multiple cloud resources. The demonstrations have been in the areas of (a) Data-Intensive Scientific Workflows on Federated Clouds, (b) Interoperability and Federation of Cloud Resources, and (c) Virtual Infrastructure Automation to enable On-Demand Services. This is a matching fund project in which Fermilab and KISTI will contribute equal resources.

## 2. INTRODUCTION

The Cloud computing paradigm has revolutionized the approach to Information Technology in many sectors of society, from telecommunication to the military to science. In particular for science, national laboratories, computing centers and universities in many countries are adapting their current paradigm on distributed computing, extending the reach for remote resource to integrate the Grid model of federated resources [1,2], the allocation-based model for national leadership-class machines, and the Cloud model of on-lease dynamically instantiated resources for private and commercial entities.

For institutions such as KISTI and Fermilab, the focus on Cloud computing is dictated by the need for more efficient support of individual large user communities (e.g. the LHC experiments) together with many medium size ones (e.g. the Intensity Frontier experiments). With each community requiring slightly different configurations for their computational environment, the use of dynamically instantiated virtual machines managed through a Cloud layer becomes an attractive solution to enable such diversity. In addition, year after year computing budgets fall short of providing the capital funds necessary to build or significantly extend new computing centers and satisfy the needs for timely access to massive resources. In reaction, more and more research institutions find it necessary to dedicate operational funds to extend the statically allocated resources under their control with Clouds to provide resource-burst capacity. In the end, to address the need for flexibility, the capability of the Cloud paradigm to instantiate computing services on-demand on a federated pool of computing hosts provides a valuable solution.

KISTI and Fermilab have been collaborating on Cloud computing since 2011. In 2013, this collaboration resulted in a formal Collaborative Research And Development Agreement (CRADA) for a multi-year program of work to offer production-quality on-demand computing services to their scientific stakeholders. The vision is to provide layered services on a federation of Clouds, provisioning execution environment on a fabric of local and remote resources, with the scientists interacting with the Software as a Service layer.

This year, Fermilab fully embraced this vision by initiating the Fermilab HEP Cloud Facility project. The Facility will integrate local resources with Grids, high-performance computing centers, and Clouds. Each resource type has its own characteristics that make it attractive for a broad range of workflow requirements. Grids are large number of statically allocated resources that follow a trust federation model; opportunistic cycles are available but never guaranteed. High-performance computers follow strict allocation processes and provide large resource capacity over a homogeneous-architecture for the allocation recipients. Clouds provide the flexibility of virtual environment with a pay-as-you-go model best suited for bursts of resource needs.

For the focus on commercial Clouds, the goals of the collaboration with KISTI have been extremely well aligned with the goals of the HEP Cloud Facility project. This report discusses how the work for this third year on Cloud federations enables effective handling of data-intensive workflows. In the years to come we envision an increase in both the diversity of resource providers and the scale of utilization, improving our ability to use Facility portals to federate resources and deploy ensembles of complex services in support scientific computation.

## 3. TECHNICAL DELIVERABLES

The technical deliverables for the CRADA project of this year had to be adjusted to reflect the focus on Cloud bursting of the HEP Cloud Facility project. Most of the deliverable described in the proposal were well aligned with the goals of the two projects and were successfully completed. The deliverables focusing on investigations to improve local Infrastructure as a Service (IaaS) Facilities and commercial Clouds, other than Amazon Web Services, have been worked at a low priority, resulting in incomplete deliverables.

The adjusted program of work for the third year of the agreement consisted in demonstrations and studies to build a production scale infrastructure to run scientific workflows on dynamically provisioned resources. The work was

organized in three major areas: (1) Data-Intensive Workflow Integration; (2) Interoperability and Federation; (3) Automation and On-Demand Services.

1. **Data-Intensive Scientific Workflows on Federated Clouds** focuses on developing and integrating mechanisms to support the execution of scientific workflows with large data processing needs. The deliverables for this area are the following:
  - a. Cost-sensitive provisioning on the AWS spot market: focus the previous studies on cost-sensitive provisioning algorithms to identify optimal bidding strategies to provision resources on the AWS spot market. Hao Wu, a PhD student from the Illinois Institute of Technology is focusing his research on this topic. The results of his research this year have been accepted at the MTAGS workshop as a paper jointly authored with KISTI [27].
  - b. Investigate execution of workflows for CERN LHC experiments: the HEP Cloud Facility is preparing to run Monte Carlo workflows on AWS and Fermilab resources at the scale of 56,000 cores (25% of global capacity) for one month in December. A demonstration at Supercomputing 2015 will show the capability of gCloud at KISTI to be integrated with the activity (see below).
  - c. Integration of RnD infrastructure with HEP Cloud Facility: run scientific workflows on federated cloud resources via GlideinWMS [4,8] and cloud web services API's. This activity demonstrates that workflows for CMS and NOvA experiment can take advantage of Cloud resources for their computational peaks through the HEP Cloud Facility infrastructure.
2. **Interoperability and Federation of Cloud Resources** consists in finding a set of virtual image formats and application programming interfaces that can be used by all members of a virtual organization across a heterogeneous infrastructure. The deliverables for this area are the following:
  - a. Improve data management by developing strategies to interact with AWS Simple Storage Service (S3) effectively: develop tools to use AWS S3 as a Storage Element fully integrated with the experiments data management services.
  - b. Demonstrate a federated Cloud between Fermilab and KISTI: a demonstration at SC2015 will show the federation capabilities between the HEP Cloud Facility at FNAL and gCloud at KISTI, running CMS Monte Carlo workflows.
  - c. VM image portability: improve AWS authentication mechanism of the automatic virtual machine image format conversion tool developed in 2014.
  - d. Perform benchmarks of Fermilab and AWS computing infrastructure: the results are used to scale the expected execution times and evaluate costs more accurately [31].
3. **Virtual Infrastructure Automation for On-demand Services** aims at finding the most efficient methods for scientific grid and cloud computing middleware to distribute data and execution across the WAN to meet the demand. The deliverables for this area are the following:
  - a. Provisioning of a platform of services: transition the mechanism to provision complicated ensembles of virtual machines in support of scientific workflows to use native AWS orchestration services (CloudFormation). This mechanism was applied to web caching services and provides automatic service discovery and scaling based on demand [31].
  - b. Improve the accounting and monitoring infrastructure: AWS provides accounting infrastructure to track cost by VM and services used; the additional infrastructure developed by this CRADA allows to compare user job execution time with overall VM running time and associate costs directly with scientific workflow execution

The following sections describe in more detail these demonstrations and studies.

### 3.1. Data-Intensive Scientific Workflows on Federated Clouds

#### *Cost-sensitive provisioning*

The work on provisioning focused on evaluating strategies to complete a scientific workflow while optimizing the cost of provisioning VM on the AWS spot market. This is an auction-based market that allows users to bid for provisioning VMs on AWS excess resource capacity. Bids target a certain instance type i.e. a given configuration of number of cores, RAM, local disk, network interfaces, etc. Successful bids provision resources for as low as one tenth of the on-demand price. The system, however, does not guarantee continuous availability, as the VM can be preempted in case of scarce excess capacity and the bid request being outbid. Different strategies balance the tradeoff between bidding low and expecting long availability of the resource to complete the workflow. The results

of this research, lead by Hao Wu, have been accepted at the MTAGS workshop as a paper jointly authored with KISTI [27].

For this work we have developed a full EC2 spot instance simulator that uses real EC2 spot pricing history to emulate the spot instance life cycle and expected charges. We reviewed eight of the most popular bidding strategies in both literature and practice and compare them in terms of cost, deadline miss rate, and task execution length for scientific workflows. The different bidding strategies were tested for jobs lasting 5, 10, and 24 hours.

Bidding strategies were categorized as static and dynamic. Static strategies bid a constant price i.e. the bid does not change as the market price changes. Examples of static strategies are bidding at a fraction (e.g. 25%) of the on-demand price, at the on-demand price or above (e.g. 10x), or at the historical minimum for a resource type. Dynamic strategies adjust the bidding prices according to application execution requirements and market prices. An example is an algorithm that finds the cheapest instance type across AWS availability zones and resource types; it then bids that price or withdraws the bid if it is below a certain threshold.

Different strategies result in different probabilities of completion (assuming to resubmit failed jobs for one week), cost (as compared to the on demand price), and overall completion time (incomplete jobs must be resubmitted and already incurred a cost). To give a sense of the results for 10 hours jobs, the strategy of bidding at 25% of the on demand price results in 48% incomplete jobs at the cost of 11% the on demand price and 1.9 days of overall execution time. For comparison, bidding at the on demand price results in 22% incomplete jobs with a cost of 20% on demand price and 1.2 days of execution time.

Our study concluded that in practice the dynamic bidding algorithms do not perform any better than the static bidding algorithms, as they are based on assumptions that do not always hold in reality. For instance, most dynamic algorithms assume that checkpointing can be performed before the instance is preempted. As for the static bidding algorithms, the one that we evaluate to have the best tradeoff between success rate, cost, and overall execution time is bidding at 25% of the on-demand price.

This study will continue to study the impact of large-scale bids, which by themselves may influence the market, and extend the evaluation to include bidding across availability zones.

#### *Investigate execution of workflows for CERN LHC experiments*

The CMS Experiment at CERN and the Fermilab Scientific Computing Division have agreed to work together on an urgent large-scale cloud computing use case for their experiment. Due to the restart of the Large Hadron Collider experiment at CERN in 2015, it is now at a higher-beam energy than previously. It will also have higher luminosity and different beam parameters, such as beam spot location. The experiment needs a large amount of initial Monte Carlo simulation and reconstruction to best utilize data from the new run. The initial proposal suggested a simulation of  $10^9$  events through the full chain of generation and simulation (GENSIM), digitization and reconstruction (DIGIRECO) in the CMS detector. This is estimated to take 56,000 compute cores running simultaneously for one month. The full simulation workflow would require fairly little input and produce an output of approximately 800TB of data. The CMS collaboration has also considered doing digitization and reconstruction of Monte Carlo that had already been generated and also reconstruction of raw data. Both of these applications take less compute time to complete and require more data transfer.

There is an agreed program of work between the Fermilab HEP Cloud Facility Project and CMS computing. CMS have responsibility for data movement to and from S3 and enhancements to their workflow management agents, as well as experiment-specific monitoring. The HEP Cloud Facility project has responsibility for procuring the Amazon services, providing the virtual machines to run the scientific workflows, the on demand services and monitoring, as well as the submission and provisioning facilities.

The early tests used a 30-minute CMS job that performed the GENSIM step on a small number of events. We were able to run this test job successfully both on Amazon Web Services and on KISTI's GCloud. As of this writing, further testing awaits a more complicated workflow to be delivered by CMS for cloud submission.

#### *Integration of RnD infrastructure with HEP Cloud Facility*

The research and development conducted by the collaboration in the past 3 years in the field of commercial clouds is fundamental to the deployment of the Fermilab HEP Cloud Facility. In particular, the experience of running scientific workflows for the NOvA experiment in 2014 set the expectations for scale in 2015 and 2016. At that time, we have demonstrated that we could run a federation of 1,000 VM simultaneously between FermiCloud and AWS. As described in the 2014 report, native OpenNebula v4.8 commands can fill the cluster with 1,000 VM in 30 minutes. FermiCloud and AWS test jobs processed a total of 20,000 Monte Carlo configuration files, reaching the limit of 1000 simultaneously running VMS (1 job/VM) on FermiCloud and 1000 simultaneously running jobs (2 jobs/VM) on AWS.

In virtue of this experience, we have prepared the HEP Cloud Facility to sustain the bursts of capacity of up to 56,000 cores for CMS. For NOvA, we have submitted and were granted a request for funding to integrate further NOvA scientific workflows with AWS. The plan is to show both the reliability of AWS as well as burst capacity. For the reliability, we plan to run multiple campaigns throughout the year and validate that AWS resources are at least as available as local Fermilab resources. For resource burst, we plan to run simulations of the NOvA near-detector, processing 60,000 configuration files in our initial trials. The integration of the NOvA submission infrastructure (jobsub) with HEP Cloud is in progress. This way, the user interface to direct scientific jobs to AWS can be initially defined as a submission command option and, eventually, as an administrative policy.

### 3.2. Interoperability and Federation of Cloud Resources

#### *Improve data management by developing strategies to interact with AWS Simple Storage Service (S3) effectively*

The collaboration has developed mechanisms for the data management systems of the experiments to use AWS S3 as a repository of data. The collaboration has conducted investigations to use AWS S3 for data input and output.

Output handling has been described in the “*data movement on-demand*” section of the year-2 report. It discusses the trade off of jobs storing output data to S3 and stream it back to archive after the job is terminated vs. sending data directly back to archive from the job. For our parameter space, considering the high-bandwidth connection with S3 and high write capacity of the archival storage at Fermilab, transferring the data back directly from the jobs is the most effective strategy.

Input handling has been analyzed in the tradeoff between staging data to S3 before jobs access it vs. transferring data directly from the archive (e.g. Fermilab storage) to the job. There is no charge to transfer data to AWS in either case. The tradeoffs for input data are similar as to output data. In the first case, data storage charges apply, while in the second virtual machines may run idle waiting for input. The optimal strategy depends on the effective bandwidth from storage, the number of jobs, the amount of data, and how long the data needs to be in S3.

Considering that direct job input has been proven to work in 2014, we developed tools for the scientific workflows to interact with S3. In particular we have (1) developed a stage-in mechanism to transfer datasets to S3 in bulk using SAM, the data handling system of the Intensity Frontier experiments and (2) integrated S3 with ifdh, the abstraction layer used by Intensity Frontier jobs to transfer data locally from storage. At a high-level, the stage-in tool transfers a dataset to S3 and declares the new location in SAM. This way, jobs processing that dataset are given the S3 location for those files by SAM and use ifdh to access them.

In order to maximize network throughput, the stage-in tool launches a number of parallel processes on different machines to initiate the upload to S3. Currently, these machines are worker nodes from the Fermilab general-purpose cluster. The tool was tested by staging in an input dataset for the NOvA experiment consisting of 18,900 files for 1.4 TB of data from the Fermilab dCache archive. The test used 40 nodes with 1 transfer process each. The maximum throughput was reached with 25 transfers, staging in the dataset in 45 minutes with an average throughput of 4.1 Gbps.

The number of processes and worker nodes should be further tuned to maximize the use of network interfaces and minimizing the total number of machines provisioned for the transfer.

#### *Demonstrate a federated Cloud between Fermilab and KISTI*

The ability to integrate resources at KISTI and Fermilab through a common Cloud management framework will be demonstrated at SuperComputing 2015 (SC15) (Austin, TX, USA – Nov 16-19, 2015). The Fermilab HEP Cloud Facility is used as the Cloud management and integration platform. Based on the glideinWMS framework, the facility can configure different end points for the provisioning of resources. It is planned that the demonstration will provide a proof-of-principle usage of approximately two hundred cores at two endpoints: AWS and gCloud, an OpenStack deployment at KISTI.

Simulation workflows for the CMS experiments will be submitted to the Fermilab HEP Cloud facility and distributed to AWS and gCloud. As virtual machines are instantiated at KISTI, the CMS software distribution will be made available through the CERN Virtual Machine File System (CVMFS), configured to rely on the Squid deployments of the CMS Tier 3 at KISTI for data caching. Jobs are envisioned to last about 30 minutes and transport results back to Fermilab storage for archiving.

Regular biweekly videoconferences throughout the year have enabled the KISTI and Fermilab teams to exchange knowledge and ideas on the project deliverables and, in particular, coordinate the preparations for the demonstration. The demonstration will rely on monitoring displays for OpenStack, the Fermilab HEP Cloud Facility, and AWS to

showcase the ability of the system to federate across community and commercial Clouds. Joint members of KISTI and Fermilab will present the demonstration at the KISTI booth at SC15.

#### *Virtual machine image portability*

At the beginning of this year we had a three-stage image creation system that served both FermiCloud and AWS images. It had been developed in the second year of the CRADA. The existing system, however, was slow in the stage that transitioned the VM from FermiCloud to AWS and also was prone to failing on various error conditions. It also was not using the current releases of the Amazon SDK.

Significant improvements that were made include the following. We switched to use HVM virtualization (hardware virtual machine under the Xen hypervisor) on AWS. This enables access to all of the resources available at Amazon, rather than the small percentage of the resources that support the paravirtualized kernel that we were previously using. We shrunk the size of the default VM from 13GB to 7 GB, thus saving import time and cost while running. We audited the steps in the import process and cleaned out several that were no longer necessary. We extended the process to cover multiple Amazon accounts and regions. We rewrote the Amazon import routine to use the Python SDK rather than the command-line utilities and implemented a more secure method of presenting credentials to the cloud using short-lived tokens. We extended the ephemeral storage detection utility to detect the presence of multiple scratch devices on a virtual machine; this way, those ephemeral storage mounts survive a reboot/restart of the virtual machine. We also added a new custom initialization script to detect the location of the on-demand squid services at launch time and modify configuration files accordingly.

The improved conversion tool is a version strengthened for production use and will be used for the operations of the Fermilab HEP Cloud Facility. With minor modifications the virtual machine images it produces are also being used at gCloud for the SC2015 demo.

#### *Perform benchmarks of Fermilab and AWS computing infrastructure*

A key piece of information in deciding when to run on the cloud is having good estimates of the relative performance of cloud virtual machines compared to local hardware. For this purpose we used two CPU benchmarks: the TTBAR and HEPSPEC06 (HS06) benchmarks. TTBAR is the stock benchmark that has been used for years by CMS to specify the performance of new physical hardware for purchase. It consists of a self-contained tarball that runs GENSIM to generate 150 top/antitop quark events. HS06 is a subset of the SPEC CPU2006 benchmark that includes only those benchmarks that are written in C++. It is a standard benchmark published by most hardware vendors. For the most part, we found that the two CPU benchmarks agree with each other in describing the performance of cloud virtual machines. Also, cloud virtual machines scale as expected i.e. if a 4-core VM and a 16-core VM of the same architecture are compared, the performance is 4 times as much.

Equally important is the bandwidth available to the AWS Simple Storage Service (S3) and from AWS to Fermilab's archive. Long transfer times translate to increased costs, as one pays for virtual machines that are waiting to transfer their files. We measured a number of reads and writes to the S3 service, looking not only for total bandwidth but also if we would get any outright failures to read. We did not observe any failures within the range we tested. As far as network bandwidth is concerned, we are using an open network topology, using normal Internet routing to get from AWS to Fermilab via the ESNNet Research network, which is peered with AWS facilities at several US points of presence. We observed up to 7Gbps of throughput from Amazon to the disks of our archive, twice what we believe is needed for our biggest CMS use case [31].

### **3.3. On-demand Services for Scientific Workflows**

#### *Provisioning of a platform of services*

In year 1 we launched individual instances of virtual machines. In year 2, we started to deploy them as an ensemble, together with the services that they rely on for scalability, focusing on Squid for web caching. For the discovery of the Squid servers at AWS, we deployed Shoal [23] at Fermilab, a naming service for Squid. In year 3, we refined the provisioning of virtual machines as ensembles by using native AWS services to automate the deployment and on-demand scaling of Squid servers [24]. This way, the Fermilab operational team does not need to maintain local infrastructure anymore.

Squid servers are deployed as virtual machines in every AWS Availability Zone (AZ) selected to execute scientific workflows. The servers are part of an "Auto-scaling Group", an AWS service that automatically deploys or removes virtual machines depending on client demand. The demand is determined by measuring the average data

throughput out of the virtual network interfaces of the Squid server VM's. When the throughput is above or below a set threshold, the Auto-scaling Group instantiates or removes Squid servers. CloudWatch, the AWS monitoring service, provides the measurement of throughput. Within an AZ, all Squid servers are available through a single "Elastic Load Balancer" (ELB). In our setting, ELB were tested to sustain 500,000 requests per minute, a rate above our requirements. The "Auto-scaling Group" updates the configuration of the ELB to maintain it up to date with the deployed Squid servers. "Route 53", the AWS Domain Name Service, defines a unique name for the ELB's in each AZ. All these services are deployed as an ensemble through "CloudFormation", the AWS service that enables the scripting of service and resource deployments.

The VM's running scientific workflows are clients of the Squid servers. They use web caching to access calibration constants through the Frontier system and scientific software through the CERN Virtual Machine File System (CVMFS). At boot time, a client VM detects the AZ in which it was deployed [26]. With this information, it configures the environment defining its web proxy to the ELB for that AZ.

Squid servers were deployed on VM of types *m3.large* and *m3.xlarge* to measure their performance. Types with more cores, memory, and disk available are more costly but can serve more clients. Considering our requirement of running CMS scientific workflows on 56,000 cores, we opted to use *m3.xlarge* types. In our load tests we scaled up to 3 of these servers, each of which served 1 Gbps of data to 16 total clients. With the deployment scheme discussed above, they are automatically scaled up as clients increase the demand. The network throughput was found to be equal to the hardware capacity of the virtual machines to serve data; the Elastic Load Balancing structure did not add any overhead.

#### *Improve the accounting and monitoring infrastructure*

To provide a comprehensive view of resource utilization, we want to track the use of Cloud resources with similar interfaces as for Grid resources. Unlike a typical Grid resource, however, access to AWS is not gated through a compute element interface, thus we cannot reuse accounting probes that track the number of batch slots used by each virtual organization. Using the earlier OpenNebula probe written for FermiCloud, we developed a similar probe to track the numbers of virtual machines launched, their start and stop times, their CPU efficiency, their termination codes, and the cost we pay for each. Data is collected once per virtual machine per hour and stored in a Usage Record similar to that used in Grid computing. This probe is written in Python and uses the BOTO toolkit to communicate with AWS.

Similar code was used for a monitoring script to collect data every 5 minutes on running VM's in the cloud and their CPU efficiency. The numbers collected are then sent to the "Graphite" monitoring server and formatted for graphical display. The display allows filtering by account number, region, availability zone, instance type, or virtual organization. There is a corresponding probe running on the HTCCondor collector/negotiator of HEP Cloud to show the number of actual virtual machines that report back to Fermilab. The Graphite server is also used to display and store historical pricing information of AWS.

## 4. QUALITATIVE AND QUANTITATIVE OUTPUT

In the past three years, the collaboration between Fermilab and KISTI has produced several results in the areas of Cloud computing. The main achievement this year consists in the integration of the techniques and methods developed for resource Cloud bursting with the Fermilab HEP Cloud Facility. The Facility will become the production infrastructure at Fermilab to interact with remote resources. For December, the Facility is preparing to run at the scale of 7,000 VM and 56,000 cores for the CMS and NOvA experiments.

#### *Quantitative considerations:*

This year the collaboration has worked on five papers. It has published one at the Workshop on Many-Task Computing on Clouds, Grids and Supercomputers (MTAGS) [27]; one at the International Symposium on Grids and Clouds (ISGC2015) [29]; two at Computing in High-Energy Physics (CHEP2015) [18, 30]; and another submitted to the IEEE International Parallel and Distributed Processing Symposium (IPDPS) [31]. In addition, the papers submitted to the journal "IEEE transactions on Cloud computing" [12, 13] in 2014 as a deliverable for the second year of the CRADA have been accepted.

In summary, the three-years program has produced 13 papers to date, adding these five to the eight published in the first two years [6, 7, 19, 9, 10, 11, 12, 13].

The work from the third year was presented at five talks [18, 27, 28, 29, 30] with one more talk submitted for June 2016 at IPDPS [31]. If that is accepted, the overall 3 years program will have been presented at 14 talks, adding these four to the ten from the first two years [10, 11, 14, 15, 16, 17, 18, 19, 20, 21].

We also made available all documentation produced [24, 25], including this report. In addition, all the code [26] is available from the code repository of the Fermilab HEP Cloud project or github.com. The code consists of the following: an improved version of the automated virtual machine image format conversion tool; the code to enable a virtual machine to detect its Availability Zone and point to the appropriate on-demand Squid service for web caching (“ondemandservices” repository); the code to send accounting and monitoring information about instantiated virtual machines and their resource usage (“monitoringaccountingbilling” repository); the code to download the history of the AWS spot price market (“spotpricehistory” repository); the scripts to benchmark the performance of computing resources and storage at AWS for the comparison with Fermilab resources (“predictionengine” repository).

*Qualitative considerations:*

Build relationships with cloud facilities in US and Pacific Rim: this year we have coordinated the joint work of KISTI and Fermilab through regular biweekly phone conferences, at which engineers and managers could express and exchange views and information on relevant topics for the collaboration. We believe that Fermilab’s experience in conducting a larger acquisition of allocations on Amazon Web Service has aspects of relevance for KISTI. This acquisition has led to obtaining substantial cost reductions applicable to research institutions. In particular, AWS has implemented the discount for outgoing network traffic through ESNet (our main research network provider) and Internet2. In addition, we have applied and were granted funding from AWS for both NOvA and CMS to bootstrap the use of Cloud computing for scientific computation at the scalability level discussed above (56,000 cores).

Optimize data path access to cloud: with the recent upgrade of one of the three AWS peering points with ESNet to 100 Gbps in Seattle, the opportunity to seamlessly integrate the Network of the National Laboratories with Commercial clouds becomes attractive. We have explored three possible solutions: using the general Internet, AWS DirectConnect, and Virtual Private Networks (VPNs) with both configurations. All solutions have different characteristics in terms of network shaping and policy. In particular, network shaping is relevant as the data egress waiver with AWS explicitly applies for traffic going mostly through ESNet and Internet2. Violations of the agreement on the traffic invalidate the waiver. Using the general Internet, the options to configure network routes is limited to expressing firewall rules to prevent traffic to locations outside of the ESNet network. This form of access is provided for free. On the other hand, DirectConnect provides sets of dedicated connections into the AWS network for a fee. This form of access enables full network routing and the hosting of address space owned by the remote institution at AWS. While this may facilitate the integration of the networks, it generates concerns related to the reuse of in-house security controls to operate remotely. These same concerns apply for VPN access. To mitigate these concerns, AWS has received FedRAMP certification, demonstrating its ability to secure its environment at the level requested by the US Government. The integration of the security controls, however, will require time; thus the Direct Connect and VPN options are not currently viable in our environment. Despite the lack of bandwidth guarantees and network shaping in the integration through the general Internet, our tests show that we can achieve a bandwidth of at least 7 Gbps and we can use firewall rules at AWS to prevent data egress waiver violations. This should be sufficient to address our use cases until at least the next fiscal year.

Identify scalability issues of cloud Application Programming Interfaces: This year we have focused on accessing the AWS interfaces to provision services and resources. We have tested the performance and reliability of the AWS Simple Storage Service (S3) in the context of our benchmarking activities without exposing any errors from the service. As observed in previous years, we are confident on the ability of AWS to scale to 1,000 VMs to address our use cases. We are now starting the scale up of the testing to the final production scale of 7,000 VMs and 56,000 cores.

Create virtual infrastructure able to interoperate with leading commercial and scientific cloud facilities: the goal of the Fermilab HEP Cloud Facility project is to provide a common user interface to a range of back end facility types. These include commercial and scientific Clouds, as well as Grid resources and High Performance Computers. While we have already demonstrated our ability to execute workflows at the scale of 1,000 VM between Fermilab and AWS in 2014, the regular use of the Facility for the NOvA and CMS use cases will also show this capability. In the context of this collaboration, we are demonstrating the integration of AWS and KISTI GCloud through the HEP Cloud Facility as a demonstration at SuperComputing 2016.

## 5. BUDGET ALLOCATION

- A. Fermilab-funded effort: TOTAL INDIRECT COSTS Fermi Research Alliance, LLC (FRA) / Fermilab FY2015 provisional indirect cost rate is currently 85.29% (Salaries – SWF), 17.09% (Travel), and 23.53% (Other Material & Services – M&S) of Modified Total Direct Cost, in accordance with Fermilab's contract with the Fermi Research Alliance, LLC (FRA) and the Department of Energy.

The budgeted amount to contribute \$100,000 of direct costs (equivalent to \$185,000 with indirect cost) was estimated at 7.2 FTE-months. The table below shows effort until the end of October.

PERSON	ADJUSTED EFFORT, FTE-Months (to Oct 31, 2015)
Garzoglio, Gabriele	0.94
Kim, Hyun Woo	3.93
Timm, Steven	3.78
<b>TOTAL</b>	<b>8.65</b>

- B. Obligated funds provided by KISTI as of Oct 2015 - Indicative report

3 IIT students for the summer (Rahul Krishnamurthy, Shivakumar Vinayagam, Hao Wu)	\$33540
1 INFN student for the summer (Davide Grassano) (labor, housing, transportation)	\$6,016
1 consultant for the summer (Eric Graubins)	\$35,200
Computing cycles at Amazon Web Services *	\$333
Travel *	\$4,906
<b>TOTAL DIRECT COST</b>	<b>\$79,996</b>
INDIRECT COST (17.09% on Travel; 23.53% on other M&S; 85.29% on SWF)	\$17,091
DOE ADMINISTRATIVE FEE (3%)	\$2,913
<b>INDICATIVE TOTAL</b>	<b>\$100,000</b>

\* Not all obligated money has yet been invoiced by the vendors or paid out. Some expenses (such as travel to KISTI) will be reconciled in November. The complete financial report will be available on December.

## 6. REFERENCES

- [1] Pordes, R. et al. (2007). *The Open Science Grid*, J. Phys. Conf. Ser. 78, 012057. doi:10.1088/1742-6596/78/1/012057.
- [2] Kranzlmüller, D., J. Marco de Lucas, and P. Öster. "The European Grid Initiative (EGI)." In *Remote Instrumentation and Virtual Laboratories*, pp. 61-66. Springer US, 2010.
- [3] Virtual machine interoperability documentation: <http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=5208>
- [4] Sfiligoi, I., Bradley, D. C., Holzman, B., Mhashilkar, P., Padhi, S. and Wurthwein, F. (2009). *The Pilot Way to Grid Resources Using glideinWMS*, 2009 WRI World Congress on Computer Science and Information Engineering, Vol. 2, pp. 428–432. doi:10.1109/CSIE.2009.950.
- [5] Seo-Young Noh, Steven C. Timm, Haeng-jin Jang: *vcluster: A Framework for Auto Scalable Virtual Cluster System in Heterogeneous Clouds*, in Cluster Computing (2013).
- [6] Hao Wu, Shangping Ren, Gabriele Garzoglio, Steven Timm, Gerard Bernabeu, Hyun Woo Kim, Keith Chadwick, Seo-Young Noh, Haeng-Jin Jang, *Automatic Cloud Bursting Under FermiCloud*, ICPADS CSS workshop, Seoul, 2013, published in IEEE Xplore Digital Library, DOI: 10.1109/ICPADS.2013.121
- [7] S. Timm, K. Chadwick, G. Garzoglio, *Grids, Clouds and Virtualization at Fermilab*, accepted in the Proceedings of the Journal of Physics: Conference Series by IOP Publishing, 2013.
- [8] P. Mhashilkar, A. Tiradani, B. Holzman, K. Larson, I. Sfiligoi, M. Rynge, *Cloud Bursting with Glideinwms: Means to satisfy ever increasing computing needs for Scientific Workflows*, accepted in the Proceedings of the Journal of Physics: Conference Series by IOP Publishing, 2013
- [9] Hao Wu, Shangping Ren, Steven Timm, Gabriele Garzoglio, Seo-Young Noh, *Overhead-Aware-Best-Fit (OABF) Resource Allocation Algorithm for Minimizing VM Launching Overhead*, 7<sup>th</sup> Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS) 2014, Nov 2014, New Orleans, Louisiana, USA
- [10] Hyunwoo Kim, Steve Timm, *X.509 Authentication/Authorization in FermiCloud*, IEEE 1<sup>st</sup> International Workshop on Cloud Federation Management, Dec 2014, London, UK

- [11] Hao Wu, Shangping Ren, Gabriele Garzoglio, Steve Timm, Gerard Bernabeu, Seo-Young Noh, *Modeling the Virtual Machine Launching Overhead under Fermicloud*, 14<sup>th</sup> IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2014), May, 2014, Chicago, IL, USA
- [12] Hao Wu, Shangping Ren, Gabriele Garzoglio, Steve Timm, Gerard Bernabeu, Keith Chadwick, Seo-Young Noh, *A Reference Model for Virtual Machine Launching Overhead*, IEEE Transactions on Cloud Computing, 2014, DOI: 10.1109/TCC.2014.2369439
- [13] Sadooghi, Iman; Hernandez Martin, Jesus; Li, Tonglin; Brandstatter, Kevin; Zhao, Yong; Maheshwari, Ketan; Raicu, Ioan; Pais Pitta de Lacerda Ruivo, Tiago; Garzoglio, Gabriele; Timm, Steven, *Understanding the Performance and Potential of Cloud Computing for Scientific Applications*, IEEE Transactions on Cloud Computing, 2015, DOI 10.1109/TCC.2015.2404821
- [14] G. Garzoglio, *On-demand Services for the Scientific Program at Fermilab*, International Symposium on Grids and Clouds 2014 (ISGC 2014), March 2014, Taipei, Taiwan
- [15] S. Timm, G. Garzoglio, *FermiCloud On-demand Services: Data-Intensive Computing on Public and Private Clouds*, HEPiX Spring 2014 Workshop, May 2014, Annecy-le-Vieux, France
- [16] S. Timm, G. Garzoglio, *FermiCloud On-demand Services: Data-Intensive Computing on Public and Private Clouds*, Computing Technique Seminar at CERN, May 2014, Geneva, Switzerland
- [17] S. Timm, *Authentication, Authorization, and Federation in OpenNebula with FermiCloud*, OpenNebula Conf 2014, Dec 2014, Berlin, Germany
- [18] S. Timm, G. Garzoglio, P. Mhashikar, J. Boyd, G. Bernabeu, N. Sharma, N. Peregonow, H. Kim, S-Y. Noh, S. Palur, *Cloud services for the Fermilab scientific stakeholders*, proceedings of Computing in High-Energy Physics 2015 (CHEP15), Apr 2015, Okinawa, Japan
- [19] Tiago Pais Pitta de Lacerda Ruivo, Gerard Bernabeu, Gabriele Garzoglio, Steve Timm, Hyunwoo Kim, Seo-Young Noh, Ioan Raicu, *Exploring Infiniband Hardware Virtualization in OpenNebula towards Efficient High-Performance Computing*, 14<sup>th</sup> IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2014), May, 2014, Chicago, IL, USA
- [20] Timm, S. (2013, Sep 23) *High Throughput and Resilient Fabric Deployments on FermiCloud*, invited talk at ISC Cloud 2013 symposium, Heidelberg, Germany.  
<https://cd-docdb.fnal.gov:440/cgi-bin/ShowDocument?docid=5202>
- [21] Timm, S. (2013, Sep. 24) *Enabling Scientific Workflows on FermiCloud using OpenNebula*, keynote talk at OpenNebulaConf 2013, Berlin, Germany. <https://cd-docdb.fnal.gov:440/cgi-bin/ShowDocument?docid=5203>
- [22] Daniel van der Ster, Arne Wiebalck, *Building an organic block storage service at CERN with Ceph*, presented at the 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP2013), pub. Journal of Physics: Conference Series 513 (2014) 042047, doi:10.1088/1742-6596/513/4/042047
- [23] Fermilab DocDB: Sandeep Palur - Squid and Shoal Server Documentation – <http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=5428>
- [24] Fermilab DocDB: Autoscaling tests of squid servers – <http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=5611>
- [25] Fermilab DocDB: Final Report – Benchmarking of cloud and local resources – <http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=5611>
- [26] Fermilab DocDB: All Code printout – <http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=5611>
- [27] H. Wu, S. Ren, S. Timm, G. Garzoglio, S. Noh, *Experimental Study of Bidding Strategies for Scientific Workflows using AWS Spot Instances*, 8<sup>th</sup> Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS) 2015, Nov 2015, Austin, Texas, USA
- [28] Timm, S. (2015, Oct. 21) *Fermilab HEPCloud Facility: Data-Intensive Computing in the Cloud* keynote talk at OpenNebulaConf 2015, Barcelona, Spain. <http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=5640>
- [29] S. Timm, G. Garzoglio, S. Fuess, G. Cooper, *Virtual Facility at Fermilab: Infrastructure and Services Expand to Public Clouds*, proceedings of the International Symposium on Grids and Clouds (ISGC2015), Mar 2015, Taipei, Taiwan
- [30] G. Garzoglio, O. Gutsche, *Diversity in Computing Technologies and Strategies for Dynamic Resource Allocation*, proceedings of Computing in High-Energy Physics 2015 (CHEP15), Apr 2015, Okinawa, Japan, invited plenary talk.
- [31] D. Grassano, S. Timm, G. Garzoglio, A. Tiradani, I. Raicu, R. Krishnamurthy, S. Vinayagam, S. Noh, *Code and Data Movement Design and Benchmarking for the Fermilab HEPCloud Facility*, submitted to 30<sup>th</sup> IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 23-27, 2016, Chicago, Illinois, USA

# Experimental Study of Bidding Strategies for Scientific Workflows using AWS Spot Instances

Hao Wu,  
Shangping Ren<sup>\*</sup>  
Illinois Institute of Technology  
10 w 31 St.  
Chicago, IL, 60616  
hwu28,ren@iit.edu

Steven Timm,  
Gabriele Garzoglio<sup>†</sup>  
Fermi National Accelerator  
Laboratory  
Batavia, IL, USA  
timm,garzoglio@fnal.gov

Seo-Young Noh<sup>‡</sup>  
National Institute of  
Supercomputing and  
Networking,  
Korea Institute of Science and  
Technology Information  
Daejeon, Korea  
rsyoung@kisti.re.kr

## ABSTRACT

Spot instance is an auction based Amazon Elastic Compute Cloud (EC2) instance provided by Amazon Web Service (AWS). It aims to help users to reduce their resource renting cost. The price for spot instances sometimes can be as low as one tenth of the price of the same type on demand instances. However, while gaining significantly cost savings on renting resources, users take risks on running instances without any availability guarantees, i.e. running spot instances can be preempted by Amazon at anytime. Spot instances that get pre-empted are not charged for their last hour and some users utilize that feature to run very short jobs. Different bidding strategies have been proposed to ensure the execution performance of tasks submitted to spot instances. In this paper, we present a full EC2 spot instance simulator that uses real EC2 spot pricing history to emulate the spot instance life cycle and expected charges. We review eight of the most popular bidding strategies in both literature and practice and compare them in terms of cost, deadline miss rate and task's execution length for scientific workflows. Our evaluation provides users a guidance on how different bidding strategies may impact the execution of scientific workflows.

## 1. INTRODUCTION

Amazon Web Service (AWS), an infrastructure-as-a-service (IaaS) cloud provided by Amazon is one of the largest public

<sup>\*</sup>The research is supported in part by NSF under grant number CAREER 0746643 and CNS 1018731.

<sup>†</sup>This work is supported by the US Department of Energy under contract number DE-AC02-07CH11359

<sup>‡</sup>This work is supported by KISTI under a joint Cooperative Research and Development Agreement CRADA-FRA 2015-0001 / KISTI-C15005.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

cloud service providers. The increasing popularity of utilizing cloud computing services is driven by two fundamental merits provided by cloud services: elastic and economic. With the elastic IaaS services, users can acquire both computing and storage resources as needed and only need to pay for the resources when they use the resources. Hence, with cloud services, users not only save monetary cost but also save the time and efforts on building computing infrastructures.

Because of these advantages, increased number of companies and organizations have started migrating their existing compute infrastructures to computer clouds. According to Google, 95% of web services are now deployed on cloud [5]. It is estimated that the cloud service market will grow to \$270B by 2020 worldwide [6]. The cloud advantages also attract researchers to utilize cloud for scientific computing. For instance, large research institutes such as Fermi National Accelerator Laboratory (Fermilab) [14], Argonne National Laboratory (ANL) [9], Brookhaven National Laboratory (BNL) [4], CERN [7] and others have begun executing scientific workflows on computer clouds.

However, there are still challenges yet to be solved for deploying scientific workflows on computer clouds. Scientific workflows are distinguished from general purpose workflows by the large amount of computing resources and execution time that it takes to complete the scientific workflow, in addition to the large amounts of data that are processed by the scientific workflow. Hence, although cloud is cost-effective in general, it can be costly for large size scientific workflows.

In later 2009, Amazon provided an auction based instance type – spot instance which allowed users to bid unused EC2 resources. Spot instances can be available at a cost as low as one tenth of the regular on-demand prices. Sometimes, users can even get free instances for short jobs. The essence behind such a huge price difference is that the service provider (Amazon) is willing to fully utilize their resources by selling spare unused resources in a relatively low price. However, while gaining significant cost reduction on renting resources, users also take risks on running instances without guarantees, i.e. spot instances may be preempted during their execution by Amazon at any time. Hence, many researchers have studied the trade-offs between cost and spot instance availability. Different bidding strategies have been proposed

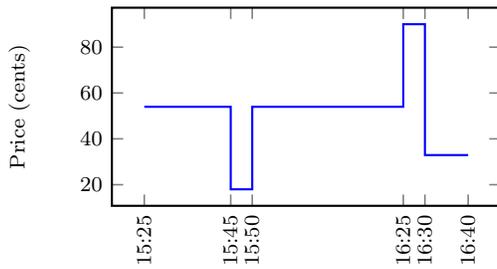


Figure 1: Spot Price Variation for m3.2xlarge Instance (2015-4-30 15:25 – 2015-4-30 16:40)

to ensure successful execution of submitted tasks and at the same time minimize of the total execution cost.

However, different bidding strategies have their own assumptions and use cases. Choosing a bidding strategy with the most balanced performance for scientific workflows is a challenging task. In this paper, we first develop a full EC2 spot instance simulator that emulates EC2 spot instance ecosystem based on real EC2 spot price history, i.e. spot instance’s life cycle and charging behaviors. We review eight of the most popular bidding strategies in the literature and from practice. We evaluate their performance in terms of cost, deadline miss rate and task’s execution length through large numbers of simulations. Our evaluation conclusions provide users a guidance on how different bidding strategies may impact the execution of scientific workflows.

The rest of paper is organized as follows: Section 2 describes the EC2 spot instance ecosystem. Different bidding strategies are reviewed in Section 3. Detailed design of our evaluation is presented in Section 4. Section 5 shows the evaluation results. At last, we conclude our work in Section 6

## 2. AWS SPOT INSTANCE

The AWS Elastic Compute Cloud (EC2) spot instance allows users to bid their own price to rent the instance. If a user wins the bid (larger or equal to market price), the user’s spot instances will be instantiated. If the user loses the bid (bid price is lower than market price), the user’s running spot instances are terminated by Amazon automatically.

### 2.1 Bidding

It is not difficult to see that the key behind the spot instance is the bidding. As long as the bid is always above market price, user’s spot instances will continue running without interruption. Although a user can bid at an extremely high price for spot instances to keep the instances running, once the market price exceeds the on-demand price, renting spot instance costs more than renting on-demand instances. In order to prevent unreasonable bid and extremely high cost, Amazon allows a maximum bid of ten times the on-demand price for each instance type [1].

There are two types of spot instances a user can bid. One is called *one time* spot instance and the other is *persistent instance*. For one time instance, the instance will be instantiated once when the bid exceeds the market price and the instance will be terminated by Amazon when the bid drops below market price or terminated by the user. Different from one time instance, for the persistent bid, spot instances remains in the system after being terminated by Amazon

and be re-instantiated once the bid surpass the market price again until they are terminated by users. It is worth noting that, for both types of spot instances, once a bid is made, it cannot be changed during the VM instance’s life time.

Amazon also support “launch group” bid, for which users can make a single bid for a group of same type spot instances. However, with the group bid, once one member of the launch group is terminated by Amazon, the entire group of the instance are terminated. In general case, Amazon only allows 20 active spot instance bids for each account in each region. For large amount of spot instance requests, user can use Spot Fleet [2] to bid spot instances which can get as much as 1000 spot instance bids per region.

### 2.2 Market Price

A bid one user made only presents the maximum price the user is willing to pay rather than the actual price the user pays for renting spot instances. The actual cost a user paid for spot instances is determined by market price. The market price is decided based on the total bids Amazon received and the size of the spot pool in a region. Amazon sorts all received bids in a decreasing order, and use the size of the spot pool to find the cut-off price. Such cut-off price is called *market price* [1]. All the bids above the cut will be granted spot instances. Once a new bid received, Amazon re-sorts the bids and then a new market price is determined.

Amazon keeps the records for all the market prices within the latest ninety days so that users are aware of recent price changes for spot instances. Such record is called Spot Pricing History, and can be retrieved from both AWS management console and APIs. Figure 1 illustrates a spot pricing history from 2015-04-30 15:25:00 to 2015-04-30 16:40:00 for m3.2xlarge instances in us-west-2a region. As shown in the figure, within the 2 hour window, the market price changed four times.

### 2.3 Charging

Amazon charges spot instance in integral hours starting from the time instance that a user’s spot instance is granted. For each integral hour, Amazon charges market spot price at the beginning of that integral hour. If the spot instance is terminated by Amazon, there is no cost for that integral hour. However, if the spot instance is terminated by user, Amazon charges the entire integral hour. Take Fig. 1 as an example. If a user bids one dollar at time 15:25, because the bid is larger than the market price (53 cents), the spot instance is granted and will start running. When the market price drops to 17 cents at 15:45, as the bid is still larger than the market price, the instance keeps running. As the price change occurs within one hour after the bid is granted, the first hour cost is based on the market price at the beginning of that hour which is 53 cents. When the instance is running for the second hour, the market price changes to 90 cents at the beginning of the second hour. Hence, the cost for the second hour is charged at 90 cents.

### 2.4 Preemption Notification

Starting from early year 2015, Amazon provides a new service to notify users the termination of their spot instances [8]. However, the notification is only available two minutes before the termination and only can be retrieved from inside the spot instance. We have not yet implemented a way to respond to the notification within the 2-minute time win-

dow.

### 3. BIDDING STRATEGIES OVERVIEW

Based on the spot instance ecosystem, it is not difficult for us to find out that different bidding strategies can result in different costs on renting spot instances. For instance, if a user bids at a low price, it is guaranteed low cost for running spot instances. However, the availability of the spot instance is not guaranteed. Many bidding strategies are proposed in the literature to balance the trade-offs between cost and availability. They can be categorized in two classes: static bidding and dynamic bidding.

#### 3.1 Static Bidding Strategy

Static bidding strategy is to bid a constant price. It does not change as the market price changes. The advantage of static bid is it is simple to implement. The drawback of the static bid is also obvious: it may not obtain any spot instance.

One typical example of using static bidding strategy is ATLAS team in BNL which always bid with one quarter of the on-demand price [10]. In this paper, we also examine other static bidding strategies such as bid with on-demand price, bid with the maximum price in the spot pricing history, bid with the absolute maximum price (ten times of on-demand price), bid with the minimum price in the spot pricing history, bid with 25% more of the minimum price in the spot pricing history.

#### 3.2 Dynamic Bidding Strategy

Different from static bidding strategy, dynamic bidding strategy dynamically adjusts bid prices according to application's execution requirements and market prices.

In 2012, Song *et al.* proposed an optimal bidding strategy for cloud service broker [12]. They first deconstruct the spot pricing history data and model the market prices using semi-Markovian chain, and formulate the problem as a cloud service broker profit maximization problem. To solve the problem, they design a profit aware dynamic bidding algorithm to calculate the optimal bid that maximizes the profit for cloud service brokers.

Song *et al.* also proposed an optimal bidding strategy for deadline constrained jobs [15]. The optimal bidding strategy calculates the bid according to the probability distribution of all the market prices existed in the spot pricing history and the remaining deadline at the beginning of each instance hour. In their paper, the authors assume that market prices are uniformly distributed. In order to meet application's deadline, once the remaining execution time equals to the deadline, the application is immediately migrated to on-demand instance.

Tang *et al.* also proposed an optimal bidding strategy for deadline constrained jobs [13]. The authors first theoretically proved that the optimal bidding strategy can be covered by a dual-option strategy: either bid with the maximum price or with zero. Then authors build a Price Transition Probability Matrix (PTPM) that records the probability of price changes from one to another in the spot pricing history. Based on the PTPM, they formulate the problem of minimizing the cost under required reliability level as a Constrained Markov Decision Process (CMDP). An optimal bid that minimizes the cost is obtained by solving the problem through linear programming, .

Recently, Bogumil *et al.* proposed an adaptive bidding strategy that minimizes the cost for renting spot instances [11]. They first find the minimal price per ECU across all instance types and availability zones from the spot pricing history. Their adaptive bidding algorithm is to find the current cheapest per ECU instances across all the instance types and availability zones. If the current lowest price is above a predefined threshold, the algorithm withdraws the bid. Otherwise, it bids with the lowest price. In addition to find the lowest bid, the adaptive algorithm also calculate the check-pointing frequency for the application.

Although some bidding strategies are claimed as optimal bidding strategies, their conclusions are based on their specific assumptions and application scenarios. In this paper, we investigate the applicability of these strategies in scientific workflow settings. In particular, we compare a set of static and dynamic bidding strategies through simulation using real Amazon spot pricing history data and real Fermilab scientific workflow requirements settings. The comparisons are performed against different evaluation criteria. The detailed simulation design and evaluation criteria are presented in next section.

### 4. AWS SPOT INSTANCE SIMULATOR DESIGN

In order to evaluate the performance of different bidding strategies, we implement a EC2 spot instance simulator that can fully emulate AWS spot instance running status and charging behavior based on real spot pricing history. The simulator is able to automatically retrieve spot pricing history from Amazon and store the data in a local database. In this manner, we can keep more than 90 days of real spot pricing data. The simulator supports both *one time* and *persistent* spot instances. The inputs of the simulator are user's bid price and bid time. The simulator calculates the instance start time, running duration and total cost. The simulator is written in Python.

#### 4.1 Simulation Design

We first give definitions for the terminologies used in the simulation design.

**Job:** we model a job as a two-tuple  $j(e, D)$ , where  $e$  is the execution time demand and  $D$  is the deadline of the job, respectively. Both execution time demand and deadline are in instance hours.

**Success Bid:** if a bid is higher or equal to the market price, it is a successful bid. Otherwise it is an unsuccessful bid. A successful bid doesn't guarantee successful execution of a task.

**Failed Execution:** if with a successful bid, a job cannot finish its execution (preempted by AWS due to price change), it is counted towards a failed execution. If with a successful bid, a job can finish its execution, but its finish time exceeds deadline, it is also counted towards a failed execution.

**Successful Execution:** if with a successful bid, a job finishes its execution before deadline, it is counted towards a successful execution.

**An Execution:** an execution  $E$  is represented as a 3-tuple  $(b, p, d)$ , where  $b$  is the bid price,  $p$  is market price of the time the bid is made and  $d$  is instance running duration with the bid. The duration  $d$  is in integral instance hours. If  $d < e$ , then  $E$  is a failed execution. If  $d \geq e$ , then  $E$  is a successful execution.

**Simulation ( $s$ ):** one simulation  $s$  is to bid spot instance to finish a job  $j$  within the job's deadline  $D$ . In one simulation, the simulator always tries to bid for instances after unsuccessful bids or failed executions. The simulation terminates when a job is successfully executed or it fails by missing deadline. The bid frequency after unsuccessful bids and failed executions is one hour.

**Failed Execution Set ( $F(s)$ ):** the failed execution set of one simulation is defined as  $F(s) = \{E_1, \dots, E_n\}$ .

**Successful Execution Set ( $S(s)$ ):** the successful execution set of one simulation is either an empty set or a set only contains one element (a successful execution).

**Deadline Miss ( $d(s)$ ):** if the successful execution set of one simulation is empty, then we say the job misses its deadline. Hence,  $d(s) = 1$  if  $S(s) = \emptyset$ . Otherwise,  $d(s) = 0$ .

**Immediate Start ( $IS(s)$ ):** if the first bid is a success bid,  $IS(s) = 1$ , otherwise,  $IS(s) = 0$ .

**Cost ( $C(s)$ ):** the total cost to execute a job includes all the cost of failed executions and the successful execution. Hence, the total cost is defined as:

$$C(s) = \sum_{i=1}^{|F(s)|} (p_i \times d_i | b_i \in F(s)) + \sum_{i=1}^{|S(s)|} (p_i \times d_i | b_i \in S(s)) \quad (1)$$

## 4.2 Evaluation Criteria

For one instance type on one U.S. availability zone, we run the single simulation every one hour based on the spot pricing history data. To evaluate the performance of different bidding strategies, we compare statistics of simulation results from two aspects. One is whether jobs finish execution within their deadlines. The other is the cost to execute the jobs. To be more specific, we define  $\text{Sim} = \{s_1, \dots, s_n\}$  as the set of simulations performed in one availability zone for a given instance type.

Immediate Start Rate (ISR) measures the possibility that a spot instance can be granted with the first bid:

$$\text{ISR} = \frac{\sum_{i=1}^n \text{IS}(s_i)}{n} \quad (2)$$

Deadline Miss Rate (DMR) measures the possibility that a job misses its deadline:

$$\text{DMR} = \frac{\sum_{i=1}^n d(s_i)}{n} \quad (3)$$

Average Cost to Demand Rate (ACDR) measures the average cost on executing a job using spot instance compared

with the cost on executing a job using on-demand instance. It is defined as follow:

$$\text{ACDR} = \frac{\sum_{i=1}^n C(s_i)}{n \times e \times p_d} \quad (4)$$

where  $p_d$  is the on-demand price for the given instance type.

Expected Execution Length to complete a job (EEL) estimates the average total execution time required to finish a job. It is defined as follow:

$$\text{EEL} = \frac{\sum_{i=1}^n (E(s_i) \times (1 - d(s_i)))}{\sum_{i=1}^n (1 - d(s_i))} \times \frac{1}{1 - D(\text{Sim})} \quad (5)$$

where  $E(s_i)$  is the execution length of  $i^{\text{th}}$  simulation.

## 5. EVALUATION RESULTS

The evaluation results are based on the simulation on almost four months real EC2 spot pricing history data from 2015-04-03 to 2015-07-29. We run simulations for all the m3 and c3 instances across all the U.S. regions and availability zones. We evaluate the performance of bidding strategies on four different jobs settings, i.e.  $j_1(5, 168)$ ,  $j_2(10, 168)$ ,  $j_3(24, 168)$  and  $j_4(100, 168)$ . Hence, we total run 324 sets of simulations. On average, each set of simulations contains more than 2000 distinguished single simulations. We evaluate eight different bidding strategies, i.e. bidding the minimum price in the spot pricing history (MinPrice), bidding 25% more of the minimum price in the spot pricing history (Min+25), bidding the maximum price in the spot pricing history (Max-Price), bidding the on-demand price (DemandPrice), bidding the absolute maximum price (Demandx10), bidding quarter of the on-demand price (Demandx.25), bidding with adaptive bidding strategy (AdaptiveBid) [11] and bidding with the optimal bidding strategy (OptimalBid) [15].

Due to the page limit, we only illustrate the evaluation results for m3.2xlarge instances on us-west-2a availability zone.

Figure 2 depicts the evaluation results for executing 5-hour jobs using m3.2xlarge instance on us-west-2a availability zone. The figure shows that no job is able to finish its execution when bidding with the minimum price in the spot pricing history. However, if we raise the bid to 25% more of the minimum price, the performance is significantly improved. Almost 75% of the jobs are able to finish before their deadlines. Bidding with quarter of on-demand price results in the similar performance as the Min+25 strategy as they result in almost at the same bidding price. If we raise the bid to the on-demand price, the job success execution rate can reach 90%. However, it will cost 9% more on all jobs' execution and 7% more on all success jobs' execution.

Since the absolute maximum bidding price is larger or equal to the maximum price exists in spot pricing history, both of them guarantee immediate start of spot instances and all the jobs finish their execution within the deadlines. However, the average cost for completing all jobs reaches 35% of the on-demand prices which is almost the double of the cost when bidding with on-demand price.

Surprisingly, the adaptive bidding strategy performs not as well as we would have expected. Almost half of the jobs failed execution with adaptive bidding. The reason that

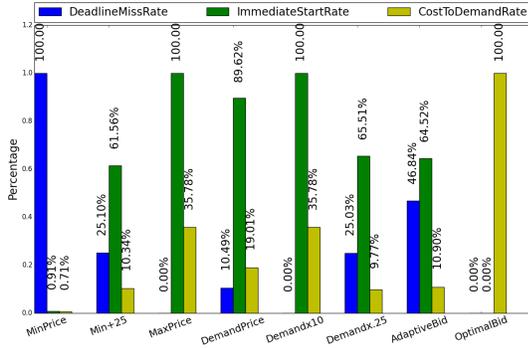


Figure 2: Performance Comparison for 5-Hour Job

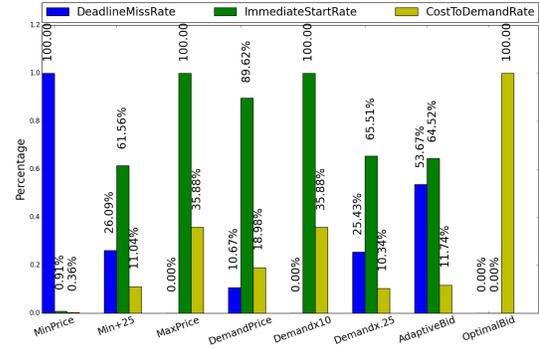


Figure 3: Performance Comparison for 10-Hour Job

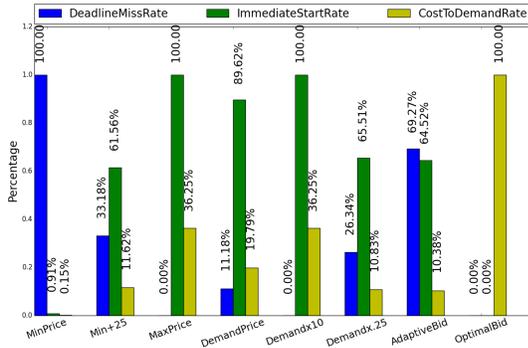


Figure 4: Performance Comparison for 24-Hour Job

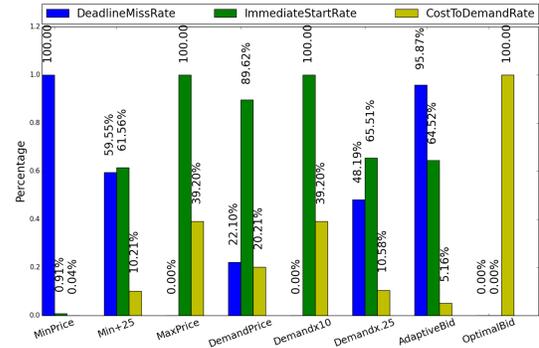


Figure 5: Performance Comparison for 100-Hour Job

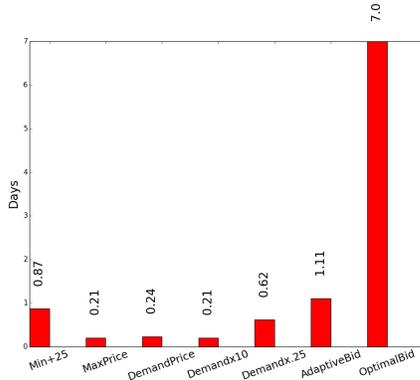


Figure 6: Average EEL for 5-Hour Jobs

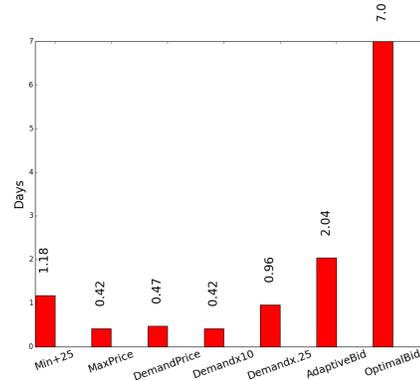


Figure 7: Average EEL for 10-Hour Jobs

adaptive bidding strategy has such a high deadline miss rate is that the adaptive bidding algorithm withdraws its bid when the calculated bid exceeds the predefined threshold. Since the adaptive bidding algorithm seeks the cheapest instance, it withdraws many bids and leads to high deadline miss rate.

With the optimal bid, all jobs are able to finish within deadline. However, if we look into the details of optimal bid, we find that the optimal bid strategy can hardly get spot instances and all the jobs are actually executed using on-demand instances. The major reason that optimal bid cannot obtain any spot instance is that the optimal bid assumes all prices in the price history are follow uniform distribution which is apparently not the case in practice.

Fig. 3, Fig. 4 and Fig. 5 show the evaluation results for ex-

ecuting 10-hour jobs, 24-hour jobs and 100-hour jobs using m3.2xlarge instance, respectively. As the execution demand increases, both deadline miss rate and the average cost for executing jobs increases. Figure 6, Fig. 7, Fig. 8 and Fig. 9 depict the average expected execution length for 5-hour jobs, 10-hour jobs, 24-hour jobs and 100-hour jobs, respectively. Since when bidding with minimum price can never get jobs to finish, we eliminate the minimum bid from the figures. By reviewing all the evaluation results, we can conclude that bidding with quarter of on-demand price gives most balanced performance in terms of cost, deadline miss rate and expected job execution length for scientific workflows.

The complete evaluation results for all m3 and c3 instances on all U.S. availability zones can be found in our website [3].

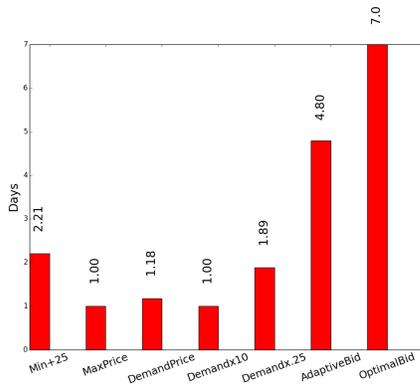


Figure 8: Average EEL for 24-Hour Jobs

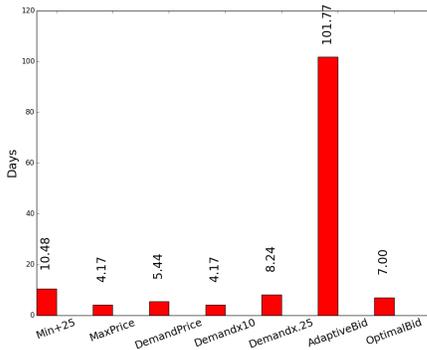


Figure 9: Average EEL for 100-Hour Jobs

## 6. DISCUSSION AND CONCLUSION

In this paper, we study the Amazon EC2 spot instance ecosystem and experimentally evaluated eight most popular bidding strategies in the literature. We develop a simulator that can fully emulate the spot instance running status and charging behavior. We use the simulator to evaluate the eight different bidding strategies in terms of job execution cost, job deadline miss rate and expected job execution length through large simulation runs. Through the evaluation, we conclude that bidding with 25% of on-demand price give most balanced performance for scientific workflows.

In addition, we also find that all the dynamic bidding algorithms do not perform as well as we would have expected. Our study reveals that these dynamic strategies are based on some assumptions which do not hold in reality. For instance the assumptions that a bid can be changed after the bid is made [15]; checkpointing can be perfectly performed before spot instances get preempted by Amazon [13]; are not valid in practice.

Another assumption that made by all the literature [15, 13, 11] is that a single bid will not affect the spot pricing. However, as previously mentioned, scientific workflows require large amounts of resources which makes the assumption invalid. Large amount of spot instance requests can have significant impact on the entire spot market. Hence, it is possible that none of the current bidding strategies works well for large amount spot instance biddings. One of our

future work is to study the impact of large scale bids and the performance of different bidding strategies on large scale bids. Since we only evaluate performance of bidding strategies on single instance types in single availability zone in our current work, another future work is to explore the bidding strategies for finding the cheapest spot instances and that are least likely to be pre-empted across multiple availability zones.

## 7. REFERENCES

- [1] AWS EC2 Spot Instance. <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html>.
- [2] AWS EC2 spot fleet. <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html>.
- [3] CODE. [code.cs.iit.edu](http://code.cs.iit.edu).
- [4] Feature - clouds make way for STAR to shine. <http://www.isgtw.org/feature/isgtw-feature-clouds-make-way-star-shine>.
- [5] Google cloud platform roadshow 2014. <https://speakerdeck.com/googlecloudplatform/keynote-cloud-developer-roadshow-2014>.
- [6] <http://www.marketresearchmedia.com/?p=839>.
- [7] Mapping the secrets of the universe with google compute engine. <https://cloud.google.com/developers/articles/mapping-the-secrets-of-the-universe-with-google-compute-engine>.
- [8] New EC2 spot instance termination notices. <https://aws.amazon.com/blogs/aws/new-ec2-spot-instance-termination-notice/>.
- [9] Nimbus and cloud computing meet STAR production demands. [http://www.hpcwire.com/hpcwire/2009-04-02/nimbus\\_and\\_cloud\\_computing\\_meet\\_star\\_production\\_demands.html](http://www.hpcwire.com/hpcwire/2009-04-02/nimbus_and_cloud_computing_meet_star_production_demands.html).
- [10] M. Ernst, C. Gamboa, J. Hover, H. Ito, and M. OConnor. Running ATLAS at scale on amazon spring 2015 HEPiX. 2015.
- [11] B. Kamiński and P. Szufel. On optimization of simulation execution on amazon ec2 spot market. *Simulation Modelling Practice and Theory*, 2015.
- [12] Y. Song, M. Zafer, and K.-W. Lee. Optimal bidding in spot instance market. In *INFOCOM, 2012 Proceedings IEEE*, pages 190–198. IEEE, 2012.
- [13] S. Tang, J. Yuan, and X.-Y. Li. Towards optimal bidding strategy for amazon EC2 cloud spot instance. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 91–98. IEEE, 2012.
- [14] S. Timm, G. Garzoglio, S. Fuess, and G. Cooper. Virtual facility at fermilab: Infrastructure and services expand to public clouds. In *The International Symposium on Grids and Clouds (ISGC)*, volume 2015, 2015.
- [15] M. Zafer, Y. Song, and K.-W. Lee. Optimal bids for spot vms in a cloud for deadline constrained jobs. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 75–82. IEEE, 2012.

# Code and Data Movement Design and Benchmarking for the Fermilab HEPCloud Facility

Steven C. Timm, Gabriele Garzoglio, Anthony  
Tiradani, Davide Grassano  
Scientific Computing Division, Fermilab  
Batavia, IL, USA

{timm,garzogli,tiradani}@fnal.gov

Rahul Krishnamurthy, Shivakumar Vinayagam, Ioan  
Raicu

Computer Science Dept.,  
Illinois Institute of Technology  
Chicago, IL

Rahul013@gmail.com,s.vinayagam15@gmail.c  
om,iraicu@iit.edu

Seo-Young Noh  
National Institute of Superconducting and Networking  
Korea Institute of Science and Technology Information  
Daejeon, Korea  
rsyoung@kisti.re.kr

**Abstract** – The nature of data that is generated by scientific experiments and the computing power required by them are often unpredictable. In order to fulfill the needs of experimenters, Fermilab has initiated a project to build the Fermilab HEPCloud Facility. This facility will enable experiments to perform the full spectrum of computing tasks, including data-intensive simulation and reconstruction, irrespective of whether the resources are local, remote, or both. It will also allow Fermilab to provision resources in a more cost-effective way, using the public cloud to provide elasticity that will allow the facility to respond to demand peaks without overprovisioning local resources. This paper describes the significant amount of preparatory work that has been done to plan and prepare the code and data movement mechanisms for the HEPCloud Facility. We have deployed a scalable caching service to deliver code and database information to jobs running on the public cloud. This uses the frontier-squid server and CVMFS clients on EC2 instances and utilizes various services provided by AWS to build the infrastructure (stack) and perform load testing on the squid servers. We have also done extensive performance benchmarking on AWS EC2 compute instances, the Amazon S3 Simple Storage Service, and the network bandwidth between Amazon and the storage elements at Fermilab to which we stage back our data. Based on the performance and cost of these services we have developed a code and data movement strategy for our experimental users.

*Scalable Infrastructure, Benchmarking, Code Distribution,  
Public Cloud Computing*

## I. INTRODUCTION TO THE HEPCLLOUD PROJECT

The Fermilab HEPCloud Facility will enable high-energy physics experiments to perform the full spectrum of computing tasks, including data intensive computing and reconstruction, using the commercial cloud as an extension of the Fermilab facility. The goal of the first year of the project is to make a facility that successfully demonstrates data-intensive computing for three key use cases. The use case for the Compact Muon Solenoid experiment at CERN (CMS) is expected to generate 800TB of output over the course of a month of running 56000 compute cores on Amazon Web Services (AWS). The use case of the NOvA experiment at Fermilab anticipates 2-3 TB both of input and output in an estimated 2 million hours of computing. The third use case is a collaboration between the Dark Energy Survey telescopic survey and the LIGO gravitational wave experiment in which the amount of computing is modest but must be done on very fast turnaround. In addition to the significant transfer of inbound and outbound data that is being processed in these high-throughput computing tasks, both applications also need to contact remote databases across the wide-area network. All of them also have very large code bases that need to be transferred to the remote cloud.

The goal of the preparatory work that has been done in this phase of the HEPCloud project is first of all to scale-test the auxiliary caching services that are used for code movement and database query caching, to be sure they can handle the expected load. We also have carefully benchmarked the compute speed, the available network

transfer bandwidth, and the throughput to the cloud-based storage system. This work is necessary so we can accurately plan the budget for these projects and estimate their total duration.

In the remainder of the paper we will describe the architecture of the scalable service, the methodology used to stress-test them, and the benchmark results. We will then describe the benchmarking work that was done on the Amazon Web Service compute instances, network bandwidth and S3 storage service components.

## II. SCALABLE SERVICES DESCRIPTION

Amazon Elastic Compute Cloud (AmazonEC2) is a web service that provides resizable computing capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay as you use.

All of the services below are provided by Amazon except the CVMFS service and the Frontier-Squid service.

### A. Elastic Load Balancer

Elastic Load Balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances in the cloud. It enables you to achieve greater levels of fault tolerance in your applications, seamlessly providing the required amount of load balancing capacity needed to distribute application traffic.

### B. Auto-scaling Groups

Auto Scaling helps you maintain application availability and allows you to scale your Amazon EC2 capacity up or down automatically according to conditions you define. You can use Auto Scaling to help ensure that you are running your desired number of Amazon EC2 instances. Auto Scaling can also automatically increase the number of Amazon EC2 instances during demand spikes to maintain performance and decrease capacity during lulls to reduce costs.

### C. Route 53

Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. It is designed to give developers and businesses an extremely reliable and cost effective way to route end users to Internet applications by connecting user requests to infrastructure running in AWS – such as Amazon EC2 instances, Elastic Load

Balancing load balancers, or Amazon S3 buckets – and can also be used to route users to infrastructure outside of AWS.

### D. CloudWatch

Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications you run on AWS. You can use Amazon CloudWatch to collect and track metrics, collect and monitor log files, and set alarms. You can use Amazon Cloud-Watch to gain system-wide visibility into resource utilization, application performance, and operational health.

### E. CloudFormation

AWS Cloud-Formation gives developers and systems administrators an easy way to create and manage a collection of related AWS resources, provisioning and updating them in an orderly and predictable fashion. We can deploy and update a template and its associated collection of resources (called a stack) in JSON or text format. It is used to setup the entire infrastructure and manage it in the same console.

### F. CERN Virtual Machine File System (CVMFS)

The CernVM File System (CernVM-FS) provides a scalable, reliable and low maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide-distributed computing infrastructure used to run data processing applications. CernVM-FS is implemented as a POSIX read-only file system in user space (a FUSE module). Files and directories are hosted on standard web servers and mounted in the universal namespace /cvmfs. Internally, it uses content-addressable storage and Merkle trees in order to maintain file data and meta-data. CernVM-FS uses outgoing HTTP connections only, thereby it avoids most of the firewall issues of other network file systems. It is actively used by small and large HEP collaborations. This is installed in the clients during boot time.

### G. Frontier-Squid

The Frontier distributed database caching system distributes data from data sources to many clients around the world. The name comes from "N Tier" where N is any number and Tiers are layers of locations of distribution. The protocol is http-based and uses a RESTful architecture which is excellent for caching and scales well. The Frontier system uses the standard web caching tool squid to cache the http objects at every site. It is ideal for applications where there are large numbers of widely distributed clients that read basically the

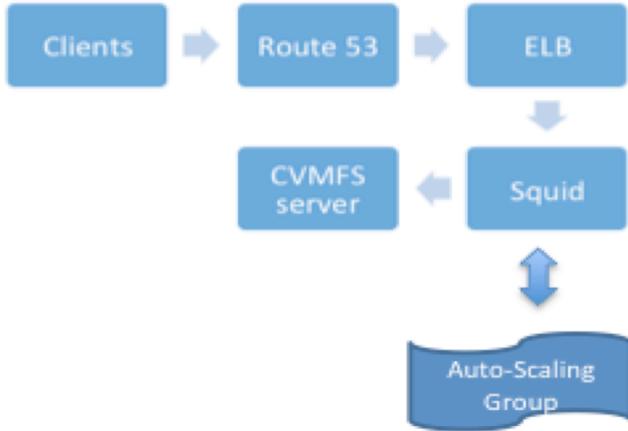


Figure 1: CVMFS Client/Server Architecture on AWS

same data at close to the same time, in much the same way that popular websites are read by many clients. The frontier-squid software package is a patched version of the standard squid http proxy cache software, pre-configured for use by the Frontier distributed database caching system.

### III. ARCHITECTURE OF SCALABLE STACK

The entire infrastructure setup of the project is done by cloud formation with the AMI's that are created. All the services that were explained work together in unison to complete the jobs given by the client/worker nodes. The architecture of the setup is given in Figure 1.

As shown in the figure the clients directly contact the URL which points to the load balancer which in turn points to the squid. The initial state of the system has only a single squid server which does most of the heavy lifting and it scales up when the network out bandwidth of the squid is more than a threshold that we set for a particular amount of time, similarly when they are idle for some time they are automatically scale down. This operation is taken care by the Autoscaling group. CloudWatch monitors all the metrics of the squid server and triggers an alarm which starts up a new instance which is a squid server. The setup will be organized in such a way that each availability zone has a similar stack and performs the same functions.

### IV. IMPLEMENTATION AND MEASUREMENT

#### A. Requirements and Installation Procedure

We use a virtual machine with Scientific Linux 6 to install the frontier-squid server. We allow 20GB for disk caching space. Squid servers are limited by network bandwidth so for these tests we compared two Amazon instance types m3.xlarge which has 4 CPU cores and an average network bandwidth of 1Gbit/sec, and m3.large which has 2 CPU cores and average network bandwidth of 700Mbit/sec. We also create client machines by installing the CVMFS client RPM as documented in the references. For these tests the clients were of instance type m3.medium. On the client machine there is a script that runs at boot time and sets the address for the squid stack based on which availability zone you are in. For example in us-west-2a availability zone the client would be automatically configured to be elb2.us-west-2a.elb.fnaldata.org, where fnaldata.org is an internal alias domain that is visible only to our virtual machines in Amazon Web Services. In production we will launch one of these service stacks in each availability zone in which we run.

We simulated the load through two different scripts. One called largequery made repeated requests (2500 in parallel) for the same 10MB file, for a total of 2.5TB of total throughput per client. Smallquery fetched a very small file a very large number of times (312500) and was designed to test the total number of requests that the load balancer can serve.

#### B. Results of load tests.

Figure 2 shows the network throughput of the three frontier-squid servers that were activated in the course of the largequery test. The three squid servers turn on one at a time as the high load continues. The full throughput of the system, including the clients, load balancers, and servers, is limited only by the maximum network throughput that the clients and servers can generate. The Elastic Load Balancer is found to not be a network traffic bottleneck. This is important because all network traffic to and from the squid servers does go through the load balancer.

Figure 3 shows the number of the requests per minute that were coming into the load balancer due to the smallquery script. It shows that the load balancer can easily handle up to 500,000 requests per minute without having any dropped requests. We observe that during periods of high load the elastic load balancer DNS entry starts to contain more IP addresses in the list of IP addresses that it returns.

We have successfully demonstrated a sustained network bandwidth that is greater than the anticipated bandwidth that will be required for database caching and code caching in our largest use case. We have demonstrated that the load balancing structure does not adversely affect network bandwidth, and that it results in no dropped requests.

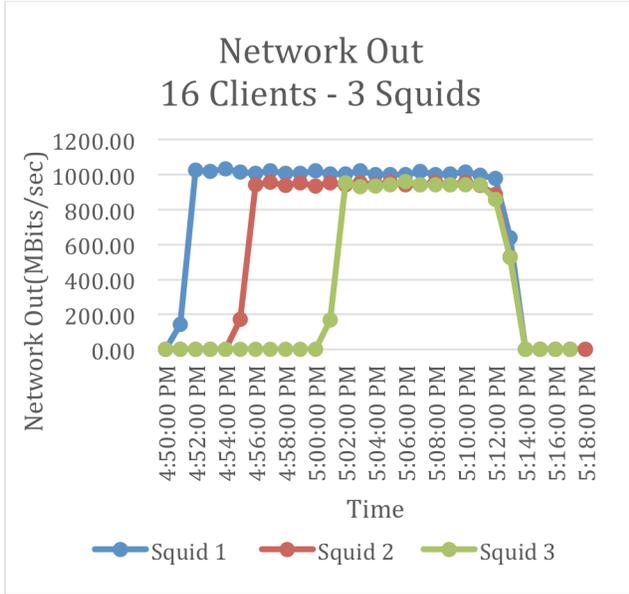


Figure 2: Network throughput of Squid server

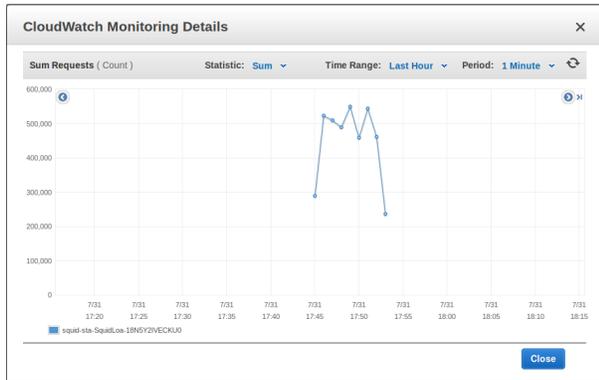


Figure 3: Load Balancer Requests per Minute

## V. BENCHMARKING DESCRIPTION

### A. Motivation for Benchmarking

When purchasing your own hardware, you would use generic and portable benchmarks, as to define the performance of the hardware at running a wide variety of tasks. When buying on-demand hardware from a third-party provider, specific benchmarks are required since the machines are bought only for the duration of a particular job, and should be the best at executing it. The study here presented regards the benchmarking of AWS instances and local cloud resources, with the purpose of using them for a full scale CMS (Compact Muon Solenoid) job. The benchmarks used were the `ttbar_gensim`, which constitute a reduced version of the first phase of the job, the `hepspec06`,

a smaller collection of packages from the more notorious SPEC2006, and some custom made bandwidth benchmarks.

### B. GENSIM Benchmarks

The `gensim` benchmark is a reduced version of what the first phase of a CMS job will be. It acts by simulating the generation of 150 `ttbar` events and storing their data by using up to 100GB.

Because of its nature, this benchmark is not only one of the most suited to assess the performances of the machine, but it also allow to monitor if the first phase of a CMS job will run smoothly without failing.

The results are given as total `ttbar/s` and `ttbar/s` per core, and can also be used to estimate the running time of a CMS job.

By running the benchmark multiple times on the same machines, it was determined that the results were very consistent, with maximum standard deviation obtained of 2%.

### C. HEPSPEC06 Benchmarks

The `hepspec06` is a subset of the SPEC benchmarks collection defined by the `all_cpp` command. The reason for choosing this benchmark lays in the fact, that the components stressed by it are the same required for a CMS job, whose code is written in C++.

Its purpose is to stress the CPU and compiler of the system, for both integer and floating point calculations and, with this being a generic benchmark, the obtained results will be more relatable, allowing for a comparison of performances with a much wider set of machines.

The results are given by the `HS06` value, which is obtained by calculating the geometric mean of the inverted ratios between the running time for each benchmark in the package and the respective associated constant. Before calculating the geometric mean, the ratios are actually averaged over 3 runs of the benchmarks, in order to obtain a statistic.

### D. Bandwidth tests

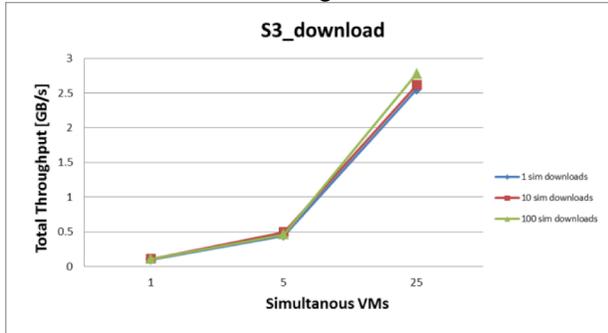
The bandwidth tests have been carried out through the usage of custom made scripts that employ the same transfer protocols and storage systems that will be adopted during the execution of a CMS job.

Amazon S3 storage is one of the possible solutions for storing intermediary files that needs to be written by the first phase of the job and read by the second phase. In order to test it, the high level `aws s3 cp` command from the AWS CLI was used to simultaneously transfer 1, 10 and 100 1GB files, from up to 25 VMs at the same time. By doing this test we hoped to determine whether we would see any outright failures of fetches from S3.

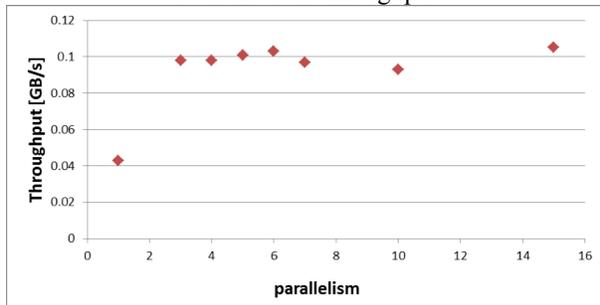
In order to store the final results of the CMS job, FermiGrid storages have been considered. The `globus-url-copy` and `xrdcp` commands were adopted respectively to transfer to 2 different servers. Due to the high latency from Amazon to Fermilab, the file transfers had to be carried out by using

multiple parallel streams, the best number of which was determined through a study of the parallelism parameter used by both commands. The globus-url-copy also allows to set the number of simultaneous TCP connection to use at the same time. With the aim of simulating the data transfer of a CMS job, 1, 5, 10 and 20 1GB files were transfer simultaneously to the storage, from up to 25 VMs at the same time.

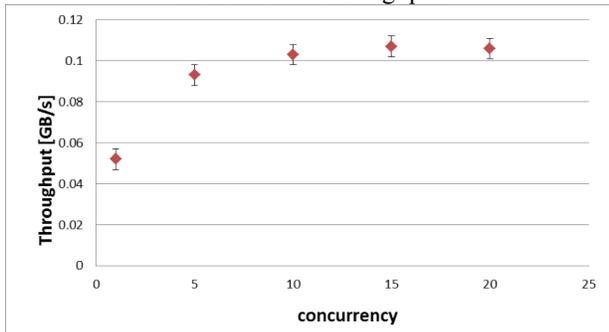
**Figure 4:** Download bandwidth throughput test from Amazon S3 to c3.2xlarge instances



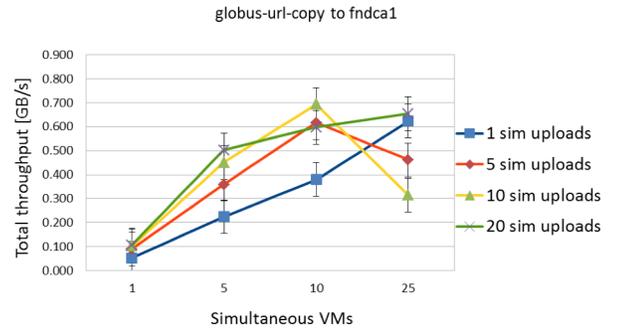
**Figure 5:** Study of the effect of the parallelism parameter over the total throughput



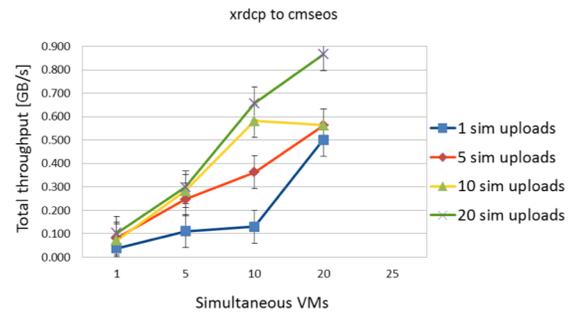
**Figure 6:** Study of the effect of the concurrency parameter over the total throughput



**Figure 7:** Total throughput analysis of the globus-url-copy command toward the fndca1 server



**Figure 8:** Total throughput analysis of the xrdcp command toward the cmseos server



Amazon	N_CORE	CORE TYPE	Speed(GHz)	\$ per hour	ttbar/s per core	ttbar/s total	ttbar per \$/h	HS06 per core	HS06 total	HS06 per \$/h
m3.xlarge	4	Xeon E5-2670	2.50	0.266	0.0139	0.0557	0.209	14.3	57.1	215
m3.2xlarge	8	Xeon E5-2670	2.50	0.532	0.0139	0.111	0.208	12.2	97.6	184
m4.xlarge	4	Xeon E5-2676	2.40	0.252	0.0201	0.0806	0.320	16.1	64.5	256
m4.2xlarge	8	Xeon E5-2676	2.40	0.504	0.0191	0.153	0.304	15.1	121	240
m4.4xlarge	16	Xeon E5-2676	2.40	1.008	0.0198	0.317	0.315	13.5	217	215
c3.xlarge	4	Xeon E5-2680	2.80	0.210	0.0153	0.0611	0.291	14.9	59.4	283
c3.2xlarge	8	Xeon E5-2680	2.80	0.420	0.0153	0.122	0.291	14.7	118	281
c3.4xlarge	16	Xeon E5-2680	2.80	0.840	0.0149	0.239	0.284	13.2	212	252
c4.xlarge	4	Xeon E5-2666	2.90	0.220	0.0228	0.091	0.415	17.5	69.9	318
c4.2xlarge	8	Xeon E5-2666	2.90	0.441	0.0226	0.181	0.410	16.5	132	300
c4.4xlarge	16	Xeon E5-2666	2.90	0.882	0.0205	0.327	0.371	14.8	237	268
r3.xlarge	4	Xeon E5-2670	2.50	0.350	0.0151	0.060	0.172	15.5	62	177
r3.2xlarge	8	Xeon E5-2670	2.50	0.700	0.0150	0.120	0.171	14.2	114	162
r3.4xlarge	16	Xeon E5-2670	2.50	1.400	0.0146	0.233	0.166	12.7	203	145
cc2.8xlarge	32	Xeon E5-2670	2.60	1.090	0.0141	0.450	0.413	11.2	359	329

Table 1: Final results from the gensim and hepspec06 benchmarks on AWS instances

bare metal	N CORE	CORE TYPE	Speed(GHz)	ttbar/s per core	ttbar/s total	HS06 per core	HS06 total
cloudworker1148	8	Intel XE ON X5355	2.66	0.0179	0.143	8.32	66.5
fnpc2036	8	AMD O pteron 2389	2.90	0.0217	0.174	12.0	96.1
fnpc3000	16	AMD O pteron 6134	2.30	0.0173	0.277	9.92	159
fnpc4001	32	AMD O pteron 6128	2.00	0.0149	0.477	8.64	277
fnpc5009	32	AMD O pteron 6134	2.30	0.0162	0.520	9.45	302
fnpc6000	64	AMD O pteron 6376	2.30	0.0136	0.873	10.0	640
fnpc7024	64	AMD O pteron 6376	2.30	0.0136	0.868	9.49	607
Fermicloud134	1	E 5-2660V2	2.20	0.0193	0.0193	17.9	17.9
Fermicloud148	1	E 5-2660V2	2.20	0.0192	0.0192	17.8	17.8
Fermicloud149	8	E 5-2660V2	2.20	0.0182	0.145	14.3	115
Fermicloud150	1	E 5640	2.60	0.0229	0.0229	18.4	18.4
Fermicloud381	8	E 5640	2.60	0.0217	0.174	15.2	122
prvm0189	1	Intel XE ON X5355	2.66	0.0173	0.0173	13.2	13.2
prvm0190	4	Intel XE ON X5355	2.66	0.0170	0.0680	11.4	45.5
FY2015 bid	48	Intel E 2670V3	2.30	0.0195	0.9381		

Table 2: Final results from the gensim and hepspec06 benchmarks on Fermilab machines

## VI. RESULTS OF BENCHMARKING

### A. TBar and GENSim

The results for the GENSIM and HEPSPC06 benchmarks are reported in Table 1 and Table 2. The cost model adopted in this analysis is based on the on-demand pricing of AWS instances, which is indicative of the ‘0.25 of the on-demand’ algorithm that is being considered for the spot market.

From the cost effectiveness alone, the best machines that have been observed would be those from the c4 and cc2 series, but this would be without taking into account that the c4s are EBS only, which means that the price of the storage

is not included in the one here presented. For this reason, the c3 instances have been considered, with particular regards for the c3.2xlarge, which comes with enough disk space, RAM and bandwidth to run a CMS job in a cost effective manner.

In order to compare local machines with the AWS ones, the same benchmarks have been run over the FermiCloud, for both VM and bare metal, obtaining the results presented in Table 2, that, when compared with those in Table 1 show that the performances of local and public cloud machines analyzed are similar.

## B. Bandwidth Test to S3

With this in mind, the study moved to the analysis of the bandwidth throughput from amazon c3.2xlarge instances to Amazon S3 and FermiGrid storage systems.

The results of the bandwidth analysis for reading from S3 are reported in Figure 4, from which it was concluded that no matter how much we would stress Amazon S3 within the capabilities of our AWS account, we would always get all the requested bandwidth, with the only limit being the maximum of 1Gbit/s per c3.2xlarge instance. We observed no error failures.

## C. Parallelism and Concurrency Analysis

Before moving to the analysis of the bandwidth to FermiGrid and CMSEOS, an analysis of the effect of the parallelism and concurrency parameters was carried out, in order to obtain the maximum efficacy for the minimum required number of inbound connections.

From the analysis of the data reported in Figures 5 and 6, it was concluded that the best solution was to set parallelism at 4 and concurrency a 5. Any values higher than this, would cause some of the uploads request to time out during the bulkier phase of the benchmarks, for what it is thought to be a problem of the dCache on the receiving server not being able to distribute all the required inbound connections.

## D. Bandwidth Test to FermiGrid and CMSEOS

Using the `globus-url-copy` command toward the `findcal` server (the general disk server for FermiGrid), and the `xrdcp` command toward the `cmseos` server (the dedicated disk storage server for CMS Tier 1), the upload bandwidth throughput from c3.2xlarge instances was analyzed.

The results reported in Figures 7 and 8 show that we were able to reach a maximum bandwidth of 5.6Gbit/s with the `globus-url-copy` and 7Gbit/s with the `xrdcp` to `cmseos`. Both of these are large storage services with multiple machines to receive the data on our end. CMSEOS has more receivers than does `findcal` so we would expect that it has slightly better throughput.

## E. Summary of Results

Through this process of CPU benchmarking, we have identified several types of Amazon instance types that will be suitable for our experimental use cases. Although the final software suite for the large production is still being finalized, we expect that the relative performance numbers between various AWS instance types and bare metal machines at Fermilab will be true to the ratios we have measured, and the relative performance numbers will be key to determining the final mix of instance types that we run.

We have also demonstrated a total network throughput from Amazon virtual machines to the Fermilab storage systems, 2.5 times larger than the expected rate of data that will be generated by the jobs in the CMS use case. Given the actual network connectivity between Fermilab and Amazon via the ESNet research network (100Gbit/s to some regions) we expect that eventually we can do even better and believe that we may currently be limited by the number of simultaneous files our server can receive at once. The combination of sufficient caching service, network throughput, storage bandwidth, and compute instances show that we are ready to analyze data in bulk on the cloud.

## ACKNOWLEDGMENT

This research is supported by the US Department of Energy under contract number DE-AC02-07CH11359.

This work is supported by KISTI under a joint Cooperative Research and Development Agreement CRADA-FRA 2015-001 / KISTI-C15005.

We acknowledge the support of the Amazon Web Services team.

## REFERENCES

- [1] S. Timm, G. Garzoglio, S. Fuess, and G. Cooper. Virtual facility at fermilab: Infrastructure and services expand to public clouds. In The International Symposium on Grids and Clouds (ISGC), volume 2015, 2015.
- [2] Squid - HTTP proxy server <http://www.squid-cache.org>, 2015
- [3] J. Blomer et al, Status and future perspectives of CernVM-FS J. Phys.: Conf. Ser. 396052013, doi:10.1088/1742-6596/396/5/052013
- [4] H. Wu, S. Ren, S. Timm, G. Garzoglio, S. Noh, "Experimental Study of Bidding Strategies For Scientific Workflows using Spot Instances." Submitted to MTAGS workshop Nov. 2015.
- [5] S. Timm et al, Cloud Services for the Fermilab Scientific Stakeholders. CHEP workshop 2015 to be published in IOP Conference Proceedings.

# Diversity in Computing Technologies and Strategies for Dynamic Resource Allocation

G. Garzoglio<sup>1</sup>, O. Gutsche<sup>1</sup>

<sup>1</sup>Fermi National Accelerator Laboratory, Batavia, IL, USA

E-mail: garzogli@fnal.gov

E-mail: gutsche@fnal.gov

**Abstract.** High Energy Physics (HEP) is a very data intensive and trivially parallelizable science discipline. HEP is probing nature at increasingly finer details requiring ever increasing computational resources to process and analyze experimental data. In this paper, we discuss how HEP provisioned resources so far using Grid technologies, how HEP is starting to include new resource providers like commercial Clouds and HPC installations, and how HEP is transparently provisioning resources at these diverse providers.

## 1. Introduction

High Energy Physics (HEP) strives to develop a detailed mathematical understanding of nature at the smallest elementary level. Its science is based on the interplay between the theory framework that describes elementary particles and elementary forces between them; and the experimental detection of particles and measurements of their interactions. It calls for probing nature at ever increasing detail to unlock the last mysteries of our universe. Also called elementary particle physics, its experimental results are based on the analysis of many individual detector measurements in comparison to corresponding simulations that are based on the current understanding of the theory. Because of this, HEP was and is traditionally a very data intensive and trivially parallelizable science discipline.

We expect that the future will see increases in number and complexity of recorded particle interactions and corresponding simulations. Using the example of the LHC [1], the second data taking period will increase the center-of-mass energy and instantaneous luminosity significantly [2]. In addition, the LHC experiments will collect a higher rate of particle interactions to maximize their physics reach [3, 4]. This translates into increasing CPU resource demands that are needed to perform the simulation and reconstruction of these particle interactions. The future is expected to bring even more increases, we will have to answer the question of how we can provide access to sufficient CPU capacity to be successful in our physics research. We call this the capacity question.

Experience from the LHC also showed that these CPU resource demands are not constant over time. They vary significantly with external triggers like for example the operation schedule of the collider, the conference schedule and vacation schedule.

As an example, Fig. 1 shows the variation of number of active analysis users of CMS over time; and the re-reconstruction passes performed in 2011 leading up to the announcement of



**Figure 1.** (*left:*) Number of CMS analysis users over time showing the variation of analysis activity, (*right:*) CMS re-reconstruction passes of data in 2011 leading up to announcement of hints for a  $\sim 125$  GeV boson at the December 2011 CERN seminar.

first hints for a  $\sim 125$  GeV boson, leading to the announcement of the Higgs Boson discovery in 2012 [5]. A computing model that adapts closely to these varying demand is generally called “elastic”. In the future, we will have to answer the question of how to introduce more elasticity into the resource allocation. We call this the elasticity question.

In this paper, we want to discuss our view on solutions for the capacity and elasticity question. We want to look at the way HEP is currently provisioning CPU resources using the Grid and look at the new technologies and providers in the form of Clouds and HPC machines.

## 2. The HEP processing challenge

In general, the HEP processing challenge is trivially parallelizable. The simulation and/or reconstruction of individual particle interactions can be treated separately. The processing of one interaction does not need input from the processing of another interaction. It is one of the best examples of the High Throughput Computing (HTC) paradigm “that focuses on the efficient execution of a large number of loosely-coupled tasks” [6].

A single batch system with access to worker nodes to execute HEP applications was sufficient in the past to realize the HTC paradigm. In most cases, these were installations hosted by universities or research institutes that handled access and support locally. With increasing demand of the community, the need to access more and more resources that are also distributed across locations and administrative boundaries arose. The Grid provided the necessary tools and services to enable easy access to a diverse group of researchers to distributed resources. Different groups of researchers are organized in virtual organizations (VOs). Computing installations at universities or research institutes joined the Grid by allowing all users of a VO to execute applications on their local resources, therefore building a trust federation of computing resources. These Grid sites defined the list of VOs that were allowed access to their local resources. Sites are pledging amounts of their resources for individual VOs, therefore formalizing resource sharing at individual sites. Pledged resources not used by a VO can be used by other VOs and are called opportunistic resources.

The Grid was and is very successful and for example enabled the LHC experiments to fulfill all computing demands for the first LHC run. It is based on batch systems which utilize directly worker nodes installed with a specific OS. Industry went a different way and used virtualization to establish a new resource sharing model: the Cloud. The Cloud replaces the batch system with a system that manages virtual machines on the physical hardware of the site and is run as a business. Commercial Clouds follow a pay-as-you-go model, where all resources are strongly

accounted and a customer pays for what was used. These business models promise near-infinity capacity and elasticity, which allows customers to use significant amounts of resources with very short ramp-up time as well as releasing them again when they are not needed anymore. In the Grid model, VOs plan their resource requests to get their work done in a defined period of time. The resource requests by VOs and the subsequent pledges by Grid sites are provisioned for peak to fulfill the VOs requirements. As shown before, the VOs demands are rarely constant over time and there are periods of lower computational demand by a VO. These free resources can be used opportunistically by other VOs following the sharing principle of the Grid. VOs that benefit from opportunistic resources themselves provide access to their unused resources at other times to the benefit of everyone. If cost effective, elasticity promised by Clouds could help in provisioning less resources permanently through the Grid and in times of demand allow for sufficient resource availability. Some commercial Cloud providers have developed in addition a spot price market, where excess unused capacity in the commercial Clouds can be given to customers at much lower prices through a bidding process. This is the Cloud equivalent to opportunistic usage of Grid sites.

A third new resource provider opening up for HEP applications are HPC installations. HPC stands for High Performance Computing and focusses on the “efficient execution of compute intensive, tightly-coupled tasks” [6]. They can, however, under certain circumstances execute HEP applications that follow the HTC paradigm. In recent time, the usage of HPC installations has become more and more accessible and feasible. HPC installations allocate resources to their users differently than traditional Grid and Cloud resources. Individual researchers or small groups of researchers are granted access to HPC installations through an allocation process. A peer review committee considers proposals designed more for individual researchers than large collaborations. In the end, allocations in time and capacity on HPC installations are awarded to successful proposals.

Table 1 shows an overview of the three resource provider types that we think will be most relevant in the near-term future to provide sufficient resource capacity and elasticity in our field.

<b>Grid</b>	<b>Cloud</b>	<b>HPC</b>
Trust Federation	Economic Model	Grant Allocation
<ul style="list-style-type: none"> <li>• Virtual Organizations (VOs) of users trusted by Grid sites</li> <li>• VOs get allocations → Pledges <ul style="list-style-type: none"> <li>– Unused allocations: opportunistic resources</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Commercial Clouds - Pay-as-you-go model <ul style="list-style-type: none"> <li>– Strongly accounted</li> <li>– Near-infinite capacity → Elasticity</li> <li>– Spot price market</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Researchers granted access to HPC installations</li> <li>• Peer review committees award Allocations <ul style="list-style-type: none"> <li>– Awards model designed for individual PIs rather than large collaborations</li> </ul> </li> </ul>

**Table 1.** Comparison of the Grid, Cloud and HPC resource provider types.

In the following, we would like to discuss the three resource provider types with emphasis on how we can use them with our HEP applications and how they can be transparently integrated into the current Grid-based setups. As the allocation models of the three provider types are rather different, we discuss how they can be integrated to support HEP needs.



- (i) The provisioning system (factories and central manager) can be shared amongst different communities and VOs.
- (ii) Separate overlay pools of resources can be provisioned per community.
- (iii) Each community has full control over their policies and priority settings within their pools.

The flexibility and ease-of-use of pilot-based submission infrastructures is important to enable the integration of Clouds and HPC installations for HEP.

#### 4. The HPC Allocation Model

HPC installations have a long history in HEP, they are used for more HPC-like applications such as Lattice QCD [10] and Accelerator Modeling [11]. Recently the interest in the user HEP communities and of the HPC installations increased to also run traditional HEP framework applications. If the HPC installation is using an Intel-based architecture, it is possible to execute HEP applications unmodified. While for non-Intel-based architectures, the cross compilation of HEP applications using native compilers is necessary. In the following section, we give examples of each of the Intel-based and non-Intel-based architecture cases.

In the Intel-based architecture case, CMS received an allocation at the San Diego Supercomputer Center (SDSC) in 2013 to re-process specific proton-proton data [12]. SDSC operates a number of Intel-based HPC clusters ranging from  $\sim 10k$  to  $\sim 50k$  cores. Individual principal investigators (PIs) submit proposals and a committee meets every three months to award allocations in the form of CPU hours. Successful proposals have one year to use the awarded allocation. Follow up proposals can be submitted. They need to demonstrate the scientific impact of the previous research. CMS took part in the allocation award procedure at SDSC with the goal to reprocess additional proton-proton data a lot faster and earlier after the LHC run 1 finished. The additional data in question had not been processed during the run itself due to processing capacity reasons and was used to publish additional physics results not reachable by the LHC run 1 data. CMS used glideinWMS pilots submitted through ssh login nodes at SDSC, processed the data, and published more than 11 papers based on the SDSC allocation. CMS is now working on follow-up proposals. As a direct reaction to the CMS/HEP use case, SDSC is preparing to give access to its HPC installations through Grid Compute Elements (CEs), making it even easier to integrate SDSC resources into pilot-based submission infrastructures.

In the non-Intel-based architecture case, Atlas was able to utilize the PowerPC-based Mira Supercomputer at Argonne National Laboratory. The machine has a similar allocation award procedure than SDSC. Proposals are required to demonstrate the ability to enable new science through the usage of Mira. Atlas cross-compiled the Alpgen event generator [13] using the IBM XL compilers for Mira and effectively ran multiple instances of Alpgen in parallel [14]. Miras almost 800k cores are subdivided into nodes and Miras minimal partition size is 512 nodes. This allows Atlas to use backfill queues to run Alpgen jobs on individual free partitions. Currently, jobs are submitted manually through a custom workflow system. In the future, the goal is to integrate Mira into the Atlas pilot-based submission infrastructure.

Both examples show that the usage of HPC installations for traditional HEP applications is possible and we can expect more usage examples in the future.

#### 5. The Cloud Allocation Model

The computing activities of experiments are not constant and, rather, follow peaks and valleys of demand as shown in Fig. 1. These are influenced by external factors, such as instrument operations, social events, conferences, holiday festivities, etc. Until recently, the only feasible approach to satisfy these peaks consisted in building computing centers at National Laboratories and Universities and procuring enough computing resources there. This spurred the creation of

resource federations and sharing agreements, embodied by Grid consortia, so that potential large available off-peak capacity could be utilized opportunistically by all members of the federation [15, 16]. As the needs for peak capacity grows, however, this strategy is becoming cost-prohibitive.

The emergence of Commercial Clouds provides a new solution to this problem. Resources have a cost only when utilized, as if they were rented rather than owned. Commercial providers offer seemingly-infinite resource capacity available on short time scales. As such, the cost of computing time is the same when renting one computing resource for 1,000 hours or 1,000 resources for one hour. There are several challenges for Cloud computing to become competitive with the Computing Centers managed by the scientific community, in terms of cost, reliability, and ease of use. Several HEP experiments and facilities, including Atlas, CMS, STAR, NOvA as well as BNL and Fermilab, are working with Cloud providers to address these challenges [17, 18, 19, 20]. Currently, the areas of work include the development of realistic economic models, resource provisioning, networking, storage, and on-demand services. We will go into more detail for all of these in the following.

### *5.1. Resource Provisioning*

Commercial Cloud providers implement proprietary application programming interfaces (API) to enable the provisioning of resource. To avoid vendor lock-in, many HEP communities rely on commonly used job management layers, such as HTCondor, to abstract access to different providers. HTCondor enables access to different Clouds by supporting the proprietary interfaces of a few Cloud Providers as well as the Amazon EC2 interface. This is a widely emulated interface that enables access to several providers, although with limitations, considering that it is not a standardized interface. This strategy makes provisioning technically possible, but does not alleviate the challenge of balancing demand for computing with cost. Two major challenges for our current technology include

- (i) the ability to expand and contract provisioned resources to control cost while the job queue is full;
- (ii) fully integrate market price-based solutions to provision Virtual Machines.

The first challenge is mainly related to policy. The priority of computational activities among scientific communities are not always straight forward. Some activities may be urgent but considered low priority. A combination of urgency and priority drives the policy to expand and contract the pool of resources to balance costs. For the second challenge, a popular example of a market-based provisioning solution is Amazon Spot pricing. The user bids the maximum price that he is willing to pay for the resource. Until the market price is below the bid, the user has access to the resource. When the market price goes above the bid, the resource is retired within a few minutes. The price varies following the demand for resources on the market. Many HEP workflows are good candidates to use Spot pricing. The Grid, in fact, implements similar preemption mechanisms when users run on opportunistic resources i.e. resources made available on the Grid, but not owned by the job submitter. On the Grid, preemption is typically implemented by the batch scheduler, which kills the job processes to make the resources available to the higher-priority job (typically a job of the resource owner). To run effectively on the Grid, most computing operations had to be made already resilient to job failure and, thus, could cope with preemption. Considering that jobs are generally submitted in bulk as part of a computing campaign, the commonly used mechanisms to achieve that include

**jobs checkpointing:** the state of the job is saved and resumed when the failed / preempted jobs are relaunched

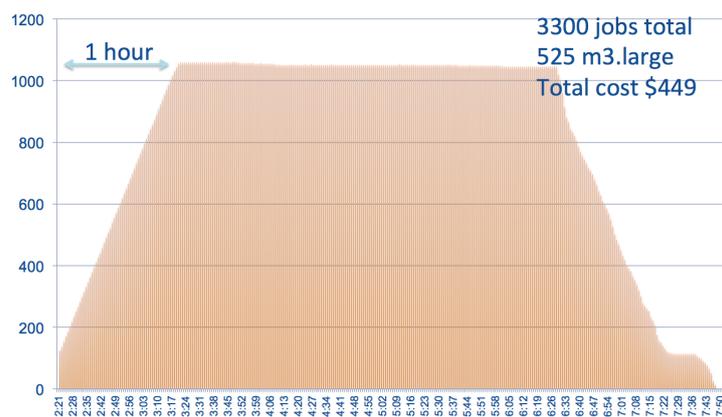
**bookkeeping:** the global state of the computation is saved through appropriate bookkeeping, so that failed jobs in a campaign can be resubmitted and the computation resumed without duplication of work (e.g. SAMWeb database of files already “consumed” in a dataset [21])

**stateless jobs:** jobs in a campaign are all equivalent to each other (e.g. some Monte Carlo production) and can be simply relaunched

**minimal unit of computation:** applications process very short units of computation (e.g. 1 event for 10 minutes), thus relaunched computations have minimal duplication (e.g. Atlas Event Service [22])

## 5.2. Economic model

With commercial Clouds becoming mainstream, computing centers at Laboratories and Universities have the choice to dynamically expand their resource pool. The decision of when to expand the pool depends on several factors, including cost. To properly manage the size of the pool, computing centers face the challenge of fully understanding their costs and compare them with the commercial providers. Preliminary cost estimates to run a “modern” computing core for one hour at National Laboratories, such as Fermilab and Brookhaven, are about \$0.03 and \$0.04 respectively [23]. For comparison, a basic virtual machine with 1 core at Amazon (m3.medium instance) cost \$0.07. The same instance, however, cost as low as less than \$0.01 using Spot pricing [24]. In addition to understanding the local cost for computing, however, predicting the costs of Commercial Cloud resources can also be a challenge. To develop an understanding of such costs, we have run computational campaigns with real physics applications on Amazon Web Services (AWS). In 2014, Fermilab has run a few Monte Carlo simulation campaigns for the NOvA experiment [25]. The largest consisted of 3,300 jobs distributed between AWS and the local Fermilab Cloud infrastructure (FermiCloud) for a scale of 1,000 jobs each (see Fig. 3). On AWS, we used dual core virtual machines (at \$0.14 / h) running two jobs per machine. The total cost was \$449, split between computational charges for \$398 and data transfers for \$51. Limiting the amount of egress data transfers e.g. by limiting auxiliary information such as log files, was key to contain that cost. Since then, however, AWS has made available to research institutions special data egress fee waivers to further reduce those costs (see Sec. 5.4).



**Figure 3.** Shown is the NOvA MC campaign running 1000 jobs in parallel on AWS.

To continue the integration of the job management infrastructure with AWS for the NOvA experiment, AWS has awarded an educational grant to Fermilab. The goal of the grant is to demonstrate the continuous availability of the resources at AWS throughout a year. We plan

to run data reconstruction for the 2014 / 2015 NOvA dataset for raw data and Monte Carlo in 16 computational campaign for a total of 2M CPU hours. As our capabilities improve, we aim to demonstrate that using the spot pricing market for this type of physics computation is cost effective and as available as the Fermilab resources.

### 5.3. Storage

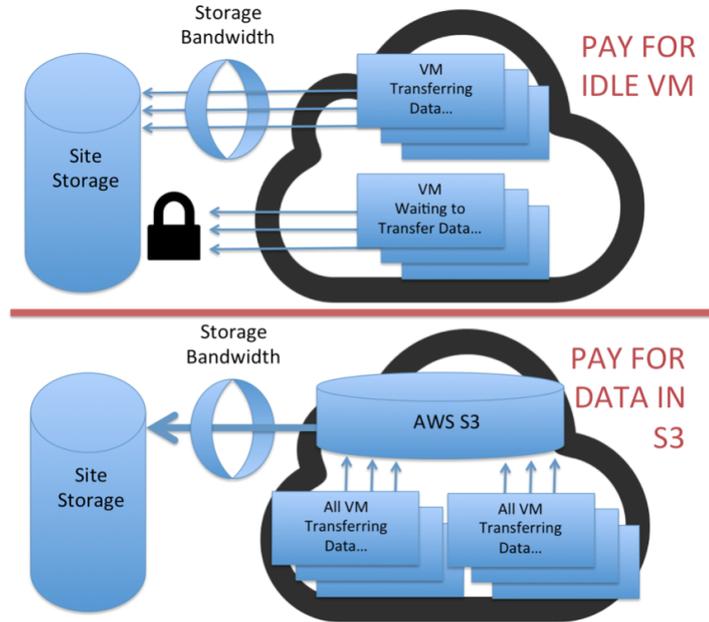
The effective utilization of compute resources depends on the effective handling of data. In general, locality of the data is known to make a difference. In particular for output data transfer, abrupt termination is a concern when provisioning resources with spot pricing. Storage locality, however, is not always more cost effective, according to our model. We consider the case where multiple jobs are submitted to the Cloud for execution and terminate approximately at the same time. We want to transfer the output data back to the home institution. We evaluate two strategies, graphically represented in Fig. 4:

- (i) Jobs attempt to transfer data directly to the remote storage at the home institution; the storage system will accept the data transfer with a certain limit on the ingress bandwidth. If data transfer is coordinated among all jobs, some jobs will transfer the data and then terminate, while others will wait in a queue. Irrespectively, virtual machines will be idle i.e. blocked on IO without running any computation for as long as the data transfer last. In addition to the data egress charges, running these idle virtual machines will contribute to the total cost.
- (ii) Jobs transfer data to local storage at AWS (Simple Storage Service - S3). Because of the high level of scalability, all jobs will be able to transfer the data at the same time using high-bandwidth. The full dataset can be transferred asynchronously directly from S3 to the home institution later on e.g. initiating the transfer from the home institution. The data egress charges will be the same as in the previous strategy. This time, however, we pay for storing the data in S3, instead of idle virtual machines.

Depending on the bandwidth available to the storage system at the home institution, the number of running VMs and the amount of data to transfer, one strategy may be more cost effective than the other. For example, Lets assume to run 1,000 jobs on 1,000 VMs of type m3.medium (\$0.07/h), each transferring 1 GB of output. The cost of data egress is \$120 irrespective of the strategy. The cost of storing the 1 TB data in S3 is generally negligible if the transfer is automatically triggered at the end of the job and then the data is erased. If the transfer is initiated manually, however, storage has a cost. For example, if it takes a week to start the transfer back to the home institution, it is about \$8. Adding related costs, such as the costs of Input / Output requests to S3, the total cost would be approximately \$132. In comparison, if we transferred the data directly from the VMs to the home institution, the cost would vary (statistically) depending on the aggregate bandwidth to storage. For example, for 20 Gbps, the cost of idle VMs would be \$8 for a total of \$128; for 2 Gbps the cost would be \$78, for a total of \$198. We are preparing to measure the cost of these strategies in realistic testbeds in the summer 2015.

### 5.4. Networking

In the Grid model, participating institutions are connected through scientific networks, such as Internet2 and ESNet in the US. These organizations absorb the cost of data transfer and, as such, this cost is hidden to the end users. This often leads to a feeling among the user community that network is a “commodity”, rather than a resource. With the transition to the Cloud model, many of these costs are exposed. Commercial Cloud providers typically allow data ingress for free, but charge for data egress and some internal data transfers [24]. Historically, however,



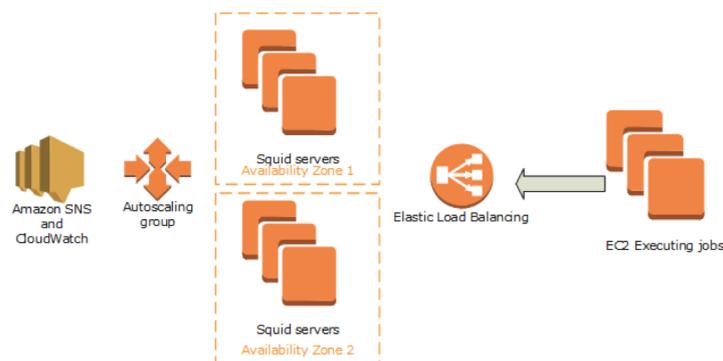
**Figure 4.** Strategies to effectively handle output data for cloud applications.

the data egress fees have acted as an economic barrier to the adoption of Cloud computing for many scientific communities. Over the past year, the scientific networks have worked to improve their network peering with AWS [26]. Absorbing much of the cost of data transfer, they are in the unique position to negotiate data egress fee discounts for the scientific community. In particular, Internet2 and ESNNet have negotiated a data egress fee waiver with AWS, by which data transfers costs below 15% of the total monthly cost are waived. As these agreements are new, some of the contractual terms are still being refined to make this an opportunity for both universities and national laboratories. Together with cost reduction, the scientific networks are working to improve the connectivity to AWS. Today ESNnet peers with AWS at three AWS zones in Seattle, Sunnyvale (CA), and Ashburn (VA). Using the default routed network, this peering allows for a connectivity of 10 GE at each point, with a planned 100 GE peering at Seattle to come in the summer. In addition to the general routed network, AWS offers a DirectConnect service, whereby network ports are reserved for certain sites. Through a pilot project, this allows for a dedicated peering of 10 GE with BNL at Ashburn and of 20 GE (2x10GE) with ESNNet at Seattle. This reserved bandwidth can be exploited by setting up dedicated circuits between the site and AWS.

### 5.5. On-demand Services

Scientific computations rely on several dependent services, such as databases, software distribution, storage, job submission queues, etc. Some of these services, such as the ones offering data caching, are known to improve the efficiency of the computation when local. As the scale of the scientific workflows running on Cloud platforms increases, the ability of instantiating dependent services also on the Cloud becomes important to improve the efficiency of the computation and, ultimately, reduce cost. We refer to these services, which are instantiated following the scale of scientific workflows that are executed on the Cloud, as on-demand services. Through our R&D programs, we have started to experiment with on-demand services such as software distribution and job submission queues. We use the CERN Virtual Machine File

System (CVMFS) [27] for software distribution. The system relies on a network of software repositories made available to remote clients through the HTTP protocol. As such, the system can scale through the adoption of web caching services, such as Squid. Our early attempts to run scientific workflows on AWS used software distribution caches at Fermilab. The lack of cache locality at AWS caused high latencies in the remote access of the software through the Wide Area Network. In addition, it caused a large number of access requests directed at Fermilab, rather than at a local cache, and overwhelmed the Fermilab distribution system. To overcome these limitations, we have developed mechanisms to elastically scale web data caching services and use them for software distribution (see Fig. 5). In short, we run a Squid server in a virtual machine at AWS. The server can be accessed through an Elastic Load Balancer, which defines a single entry point to the data caching system for the clients. The network traffic on the Squid VM is monitored through an AWS service called “Autoscaling Group”. As the traffic increases because of demand, the autoscaling group can elastically instantiate additional Squid servers. These, after their cache is loaded, enable the automatic scaling of the data distribution service. In addition, the autoscaling group can retire Squid servers as the load to the system decreases below a set threshold.



**Figure 5.** Mechanisms to elastically scale a squid web data caching service in an AWS service called “Autoscaling Group”.

Since web data caching is a service with a limited, generally disposable, state, the automated scaling of the service is relatively straightforward. More care had to be taken for the automated scaling of job submission queues. In particular, the scaling down of the service required for the system to wait the draining of the user jobs, a process that may take days. This is an active area of R&D.

## 6. Virtual Facility

The elasticity promised by commercial Cloud providers can not only be used to the benefit of VOs or science communities. Also traditional Grid sites can benefit from it.

In what we call the virtual facility approach, a Grid site would not provision anymore all needed resources through physical hardware. That hardware would need to be operated and maintained in the sites own data centers. Sites could fulfill their users needs through a combination of owned and rented resources, therefore alleviating the effect of having to provision for peak demand and rather be more elastic and cost effective. Sites would develop a cost model for physical resources and commercial Cloud resources and would optimize costs by choosing a balance between them. The agreement between users and sites about service levels of resources would stay the same. The site, however, would need to make sure that their usage

of Cloud resources would yield in the same service levels as their own physical resources. This would include investigating storage and on-demand auto-scaling service solutions for Clouds as discussed above. In the end, sites could provide complete solutions for their users with their jobs running transparently on physical or rented hardware, while optimizing costs for the sites.

## 7. Community Solutions

We have discussed three different resource providers and how they can be integrated to run HEP applications. Utilizing these providers efficiently and at high scale however requires technical knowledge and effort. Large VOs, such as the LHC VOs, have their own teams of experts that take care of integration and operations. Not every VO, however, can afford this level of sophistication. To address this limitation, organizations have been funded to provide the community at large, even beyond HEP, with the capabilities, services, and infrastructure to execute their applications at scale on multiple resource providers. One such organization is the Open Science Grid (OSG) [15].

The Open Science Grid was initially founded with the goal to share the infrastructure of the LHC experiments and other Experiments, Universities, and Laboratories in the US. From the beginning, the emphasis was to include scientific communities beyond HEP to transfer the expertise of the LHC experiments to run HTC applications at high scale to multiple scientific disciplines. The community effort is based on the premise that resource owners want to share their resources to maximize the benefit to all without relinquishing control of their local resources. Major clusters at Universities and National Laboratories connect to the OSG and control the sharing policies locally.

One goal of OSG is that researchers use a single interface to all kind of resources: resources they own; resources others are willing to share; resources that they have an allocation on (for example HPC installations); resources they buy from a commercial (Cloud) provider. OSG focuses on making this technically possible.

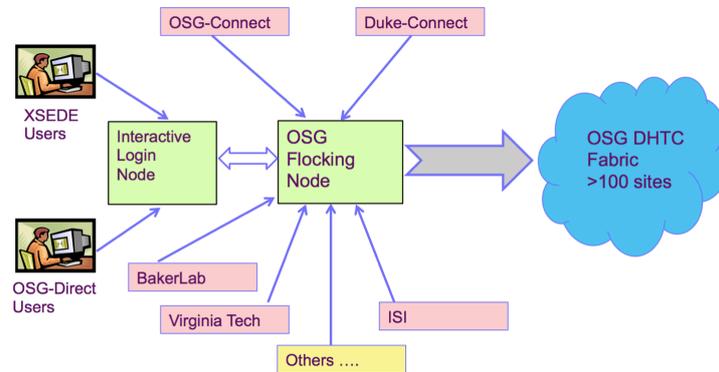
OSG operates a shared production infrastructure, called the Open Facility. It is based on glideinWMS and enables researchers to easily and efficiently run on different resource providers. OSG also maintains and advances a shared software infrastructure, called the Open Software Stack. It enables researchers to use common tools and techniques to execute their applications at scale on the OSG. In addition, OSG takes care of documentation and training of technologies and techniques to spread the knowledge across researchers, IT professionals, and software developers, creating an Open Ecosystem all research groups to benefit from the advances of the distributed high scale HTC model.

Fig. 6 shows a schematic setup of the OSG Open Facility, where different user and user groups are provided with facilities tailored to their needs to connect to the OSG.

Single Principal Investigators (PIs) can benefit from the OSG Connect service, whereby OSG operates a login node for the researcher and provides disk space and a software repository. Through the common submission infrastructure, OSG assists the PI to provision resources across the OSG facilities. OSG maintains also a dedicated instance of the OSG Connect service to serve the resource needs of researchers from the HPC community. They are awarded allocations on OSG through the XRAC process of XSEDE [28].

Universities and laboratories that are connected to the OSG have the possibility to also benefit from unused capacity at other OSG facilities by moving excess local load to the OSG, as well through HTCondor and glideinWMS, therefore virtually expanding their local resources.

LHC experiments and other large VOs use the OSG directly by operating OSG sites and using them through their own submission infrastructures, but gaining access to other OSG facilities as well.



**Figure 6.** Schematic setup of the OSG Open Facility.

## 8. Resource Allocation Models

All three presented resource provider types have very different resource allocation models. The Grid allocates resources through pledges given to VOs at sites. These pledges are constant over time and usually given for a year at a time and then renewed. Commercial Clouds follow an economic model where users pay only for what they use. There is no predefined time structure, provisioning 1 CPU for 1000 days costs the same as 1000 CPUs for 1 day. HPC installations grant allocations on their facilities in the form of CPU time that can be used in a given time frame. All three allocation models have different time frames and different mechanisms of defining the amount of resources allocation (Grid: job slots, Cloud: cores, HPC: CPU time). Although still in its infancy, the integration of these allocation models would simplify the operations of the composite cyberinfrastructure. We don't have an immediate solution on how to seamlessly integrate these resource providers and also newer ones that have not been mentioned here, but we think it is important to bring up the issue and pose the question to the community to start the discussion and develop solutions on how to combine these resource allocation models.

## 9. Summary and Outlook

In this paper, we discuss how the resource usage for HEP and other sciences is changing to include more types of resource providers. The Grid is being augmented by commercial Clouds and HPC providers. Service developed for the Grid, such as workload management systems, are enhanced to integrate the new resource providers through pilot-based submission systems like glideinWMS. The integration of commercial Clouds poses challenges in several areas. As these are addressed, we envision that Clouds will provide an ever larger fraction of the resource pool through the use of cost competitive models, such as the spot market price.

HPC installations are currently used to solve specific problems in HEP computing. As the community develops more experience in the operations of HPC, we envision growth opportunities in the resource pool from this provider type. We discussed the concept of the virtual facility combining owned and rented resources to optimize costs and provide more elasticity for the users. We think that this concept has several benefits for a facility and we expect to hear reports from implementations and modifications to the concept in the future. We also discussed a community solution based on the Open Science Grid, which enables the whole community from individual researchers to large VOs to benefit from the advances in distributed large scale HTC application execution. We think the approach of the OSG is an excellent example how the advances coming from the Grid world combined with new resource providers can be easily utilized by a larger community. In the end, we discussed that although we can use a variety of resource providers transparently through our submission infrastructures, the allocation model

are sufficiently different that new solutions need to be found for a tighter integration.

## 10. Acknowledgements

We would like to thank the various funding agencies from all over the world that made the research discussed here possible, particularly the Department of Energy in the United States. Many thanks to all our colleagues who helped gathering and organizing information and for fruitful discussions, especially Stuart Fuess, Burt Holzman, John Hover, Bo Jayatilaka, Jim Kowalkowski, Ruth Pordes, Panagiotis Spentzouris, Steve Timm, Margaret Votava, Frank Würthwein.

## References

- [1] Evans L and Bryant P 2008 Lhc machine *Journal of Instrumentation* **3** S08001 URL <http://stacks.iop.org/1748-0221/3/i=08/a=S08001>
- [2] CERN 2014 *5th Evian Workshop on LHC beam operation* (Geneva: CERN) organisers: Lamont, M; Meddahi, M; Goddard, B URL <https://cds.cern.ch/record/1968515>
- [3] et al S A 2015 Upgrade of the atlas central trigger for lhc run-2 *Journal of Instrumentation* **10** C02030 URL <http://stacks.iop.org/1748-0221/10/i=02/a=C02030>
- [4] Bawej T A e a 2014 The New CMS DAQ System for Run 2 of the LHC Tech. Rep. CMS-CR-2014-082 CERN Geneva URL <https://cds.cern.ch/record/1711011>
- [5] Chatrchyan S *et al.* (CMS) 2012 Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC *Phys.Lett.* **B716** 30–61
- [6] EGI Glossary: [https://wiki.egi.eu/wiki/Glossary\\_V1](https://wiki.egi.eu/wiki/Glossary_V1)
- [7] Bird I, Bos K, Brook N, Duellmann D, Eck C *et al.* 2005 LHC computing Grid. Technical design report
- [8] Sfiligoi I, Bradley D C, Holzman B, Mhashilkar P, Padhi S and Wurthwein F 2009 The pilot way to grid resources using glideinwms *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 02* CSIE '09 (Washington, DC, USA: IEEE Computer Society) pp 428–432 ISBN 978-0-7695-3507-4 URL <http://dx.doi.org/10.1109/CSIE.2009.950>
- [9] Thain D, Tannenbaum T and Livny M 2005 Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience* **17** 323–356
- [10] Blum T, Van de Water R, Holmgren D, Brower R, Catterall S *et al.* 2013 Working Group Report: Lattice Field Theory (*Preprint* 1310.6087)
- [11] Amundson J, Macridin A and Spentzouris P 2014 High Performance Computing Modeling Advances Accelerator Science for High Energy Physics *IEEE Comput.Sci.Eng.* **16** 32–41
- [12] Press Release: SDSCs Gordon Supercomputer Assists in Crunching Large Hadron Collider Data [http://ucsdnews.ucsd.edu/pressrelease/sdscs\\_gordon\\_supercomputer\\_assists\\_in\\_crunching\\_large\\_hadron\\_collider\\_data](http://ucsdnews.ucsd.edu/pressrelease/sdscs_gordon_supercomputer_assists_in_crunching_large_hadron_collider_data)
- [13] Mangano M L, Moretti M, Piccinini F, Pittau R and Polosa A D 2003 ALPGEN, a generator for hard multiparton processes in hadronic collisions *JHEP* **07** 001
- [14] Childers T, Le Compte T, Uram T and Benjamin D 2015 Simulation of lhc events on a million threads *21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015)* URL <https://indico.cern.ch/event/304944/>
- [15] Pordes R, Petravick D, Kramer B, Olson D, Livny M, Roy A, Avery P, Blackburn K, Wenaus T, Wrthwein F, Foster I, Gardner R, Wilde M, Blatecky A, McGee J and Quick R 2007 The open science grid *Journal of Physics: Conference Series* **78** 012057 URL <http://stacks.iop.org/1742-6596/78/i=1/a=012057>
- [16] Kranzlmüller D, de Lucas J and ster P 2010 *Remote Instrumentation and Virtual Laboratories* ed Davoli F, Meyer N, Pugliese R and Zappatore S (Springer US) pp 61–66 ISBN 978-1-4419-5595-1 URL [http://dx.doi.org/10.1007/978-1-4419-5597-5\\_6](http://dx.doi.org/10.1007/978-1-4419-5597-5_6)
- [17] Timm S, Garzoglio G *et al.* 2015 Cloud services for the fermilab scientific stakeholders *21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015)* URL <https://indico.cern.ch/event/304944/session/7/contribution/448>
- [18] Taylor R *et al.* 2015 Evolution of cloud computing in atlas *21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015)* URL <https://indico.cern.ch/event/304944/session/7/contribution/146>
- [19] Balewski J, Lauret J, Olson D, Sakrejda I, Arkhipkin D, Bresnahan J, Keahey K, Porter J, Stevens J and Walker M 2012 Offloading peak processing to virtual farm by star experiment at rhic *Journal of Physics: Conference Series* **368** 012011 URL <http://stacks.iop.org/1742-6596/368/i=1/a=012011>

- [20] Colling D *et al.* 2015 The diverse use of clouds by cms *21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015)* URL <https://indico.cern.ch/event/304944/session/7/contribution/230>
- [21] Mengel M, Norman A *et al.* 2015 Replacing the engines without stopping the train; how a production data handling system was re-engineered and replaced without anyone noticing *21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015)* URL <https://indico.cern.ch/event/304944/session/4/contribution/463>
- [22] Wenaus T *et al.* 2015 The atlas event service: A new approach to event processing *21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015)* URL <https://indico.cern.ch/event/304944/session/4/contribution/463>
- [23] Ernst M, Hogue R, Hollowell C, Strecker-Kellog W, Wong A and Zaytsev A 2014 Operating dedicated data centers is it cost-effective? *Journal of Physics: Conference Series* **513** 062053 URL <http://stacks.iop.org/1742-6596/513/i=6/a=062053>
- [24] Amazon EC2 Pricing <http://aws.amazon.com/ec2/pricing/>
- [25] Norman A 2015 Large scale monte carlo simulation of neutrino interactions using the open science grid and commercial clouds *21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015)* URL <https://indico.cern.ch/event/304944/session/9/contribution/465>
- [26] Hover J 2015 Running atlas at scale on amazon ec2 *HEPiX Spring 2015 Workshop* URL <https://indico.cern.ch/event/346931/session/9/contribution/20>
- [27] Blomer J, Buncic P, Charalampidis I, Harutyunyan A, Larsen D, and Meusel R 2012 Status and future perspectives of cernvm-fs *Journal of Physics: Conference Series* **396** 052013 URL <http://stacks.iop.org/1742-6596/396/i=5/a=052013>
- [28] Towns J, Cockerill T, Dahan M, Foster I, Gaither K, Grimshaw A, Hazlewood V, Lathrop S, Lifka D, Peterson G D, Roskies R, Scott J R and Wilkens-Diehr N 2014 Xsede: Accelerating scientific discovery *Computing in Science and Engineering* **16** 62–74 ISSN 1521-9615

# Cloud services for the Fermilab scientific stakeholders

S Timm<sup>1</sup>, G Garzoglio<sup>1</sup>, P Mhashilkar<sup>1\*</sup>, J Boyd<sup>1</sup>, G Bernabeu<sup>1</sup>, N Sharma<sup>1</sup>, N Peregonow<sup>1</sup>, H Kim<sup>1</sup>, S Noh<sup>2</sup>, S Palur<sup>3</sup>, and I Raicu<sup>3</sup>

<sup>1</sup>Scientific Computing Division, Fermi National Accelerator Laboratory

<sup>2</sup>Global Science experimental Data hub Center, Korea Institute of Science and Technology Information

<sup>3</sup>Department of Computer Science, Illinois Institute of Technology

E-mail: {timm, garzoglio, parag, boyd, gerard1, neha, njp, hyunwoo }@fnal.gov, rsyong@kisti.re.kr, psandeep@hawk.iit.edu, iraicu@cs.iit.edu

**Abstract.** As part of the Fermilab/KISTI cooperative research project, Fermilab has successfully run an experimental simulation workflow at scale on a federation of Amazon Web Services (AWS), FermiCloud, and local FermiGrid resources. We used the CernVM-FS (CVMFS) file system to deliver the application software. We established Squid caching servers in AWS as well, using the Shoal system to let each individual virtual machine find the closest squid server. We also developed an automatic virtual machine conversion system so that we could transition virtual machines made on FermiCloud to Amazon Web Services. We used this system to successfully run a cosmic ray simulation of the NOvA detector at Fermilab, making use of both AWS spot pricing and network bandwidth discounts to minimize the cost. On FermiCloud we also were able to run the workflow at the scale of 1000 virtual machines, using a private network routable inside of Fermilab. We present in detail the technological improvements that were used to make this work a reality.

## 1. Introduction

The Fermilab scientific program includes several running experiments, both the CMS experiment at the Energy Frontier, and the various neutrino and muon experiments on the Intensity Frontier. The ongoing data analysis and simulation for running experiments, combined with a large simulation load for future facilities and experiments, results in an unprecedented level of computing demand. This paper describes recent progress in the ongoing program of work to expand our computing to the distributed resources of grids and public clouds.

As part of the joint collaboration between Fermilab and KISTI, we have a program of work building towards distributed federated clouds. In the summer of 2014 the primary goal of this program was to demonstrate a federated cloud running at the 1000 Virtual Machine scale, using our local private cloud nodes and Amazon Web Services EC2. Our application of choice for this is the Cosmic Ray simulation of the NOvA experiment far detector at Ash River [9]. This application

---

\* To whom any correspondence should be addressed.

requires negligible input and produces about 250MB of output per job, and is quite computationally intensive. The NOvA experimenters spent considerable effort in optimizing their code to run at various sites outside of Fermilab, including loading their code into the OSG OASIS CVMFS server. The NOvA experiment supplied us with a set of files and scripts, which would generate one full set of their cosmic ray Monte Carlo, 20000 input files in all, with one job per file.

## **2. Challenges in using cloud resources at large scale**

The main challenge that had to be addressed in expanding to larger scale on the public cloud was finding a scalable way to distribute the code to 1000 simultaneous jobs. We also needed the capacity to quickly convert a local virtual image to Amazon Web Services format so we could make changes in the setup as needed. On the private cloud we had to find a faster and more scalable way to deliver virtual machine images, and find a more scalable virtual machine instantiation API, as well as add a significant number of machines to our private cloud. We were able to leverage existing and well-known tools for these requirements with minor modifications and adjustments.

### *2.1. Description of Squid Caching and Shoal Discovery Services*

The CERN Virtual Machine File System (CVMFS) [2] is widely adopted by the High Energy Physics (HEP) community for the distribution of project software. CVMFS is a read-only network file system that provides access to files from a CVMFS Server over HTTP. Though initially developed for virtual machines it is also used in non-virtualized environments as well. When CVMFS is used on a cluster of worker nodes, the Squid HTTP web proxy can be used to cache the file system contents, so that all subsequent requests for that file will be delivered from the local HTTP proxy server. Typically, a HEP computing site has a local or regional Squid HTTP web proxy [3], with the central CVMFS servers located at the main laboratory, such as CERN for the LHC experiments. In a remote cloud deployment we have found it necessary both for security reasons and for network latency reasons to co-locate the squid web proxies in the cloud.

In IaaS cloud resources the compute nodes and the squid caching proxies are launched dynamically with addresses that are not predictable in advance. We need to identify new methods to enable the compute nodes to dynamically discover the Squid services and other services which may be needed, and reconfigure the compute nodes to use these services. The Shoal mechanism was developed at the University of Victoria for these purposes [4,5]. We use Shoal as a service that can dynamically publish and advertise the available Squid servers. Shoal is ideal for an environment using both static and dynamic Squid servers, and distributed across multiple locations including on-site machines as well as commercial clouds.

Shoal is divided into three logical modules, a server, an agent, and a client, and is available via the python package index. The Shoal Server maintains a list of active Squid servers in volatile memory and receives AMQP messages from them. It provides a RESTful interface for Shoal Clients to retrieve a list of geographically closest Squid servers. It provides a web user interface to easily view Squid servers being tracked. The Shoal Agent is a daemon that runs on Squid servers to send an Advanced Message Query Protocol (AMQP) [6] message to Shoal Server on a set interval. Every Squid server wishing to publish its existence runs Shoal Agent on boot. Shoal Agent sends periodic heartbeat messages to the Shoal Server (typically every 30 seconds). The Shoal Client is used by worker nodes to query the Shoal Server to retrieve a list of geographically nearest Squid servers and adjust the configuration of the worker node appropriately. Shoal Client is designed to be simple (less than 100 lines of Python) with no dependencies beyond a standard Python installation. AMQP forms

the communications backbone of Shoal Server. All information exchanges between Shoal Agent (Squid Servers) and Shoal Server are done using this protocol, and all messages are routed through a RabbitMQ [7,8] Server.

## 2.2. Design and Implementation of Squid Service in the Cloud

We deployed the software stack of CVMFS (network file system), Squid (on-demand caching service) and Shoal (squid cache publishing and advertising tool designed to work in fast changing environments) on FermiCloud (private cloud) and on Amazon Web Services (Public cloud) using Serverless Puppet (Puppet apply). As a part of this work, we developed Puppet modules and scripts for installing and deploying Shoal client, agent and server. We also modified Shoal Server such that it could publish Squid Servers running on EC2 instances where the Squid port is only visible on the internal AWS Virtual Private Cloud network, and contributed this change back to the upstream developers.

The architecture of large scale batch of dynamically instantiated FermiCloud and EC2 worker nodes provided with network file system, on-demand caching service and a cache publishing and advertising tool is shown in Figure 1.

When a new Shoal Server node is instantiated, the Shoal Server and its dependent components including Apache and RabbitMQ server are installed on startup of the machine. For this purpose we constructed a Puppet script which can be run at boot time from a normal Puppet server, or as a standalone script with Puppet apply, using the cloud-init features of AWS or the contextualization features of OpenNebula respectively.

When a Squid Server is instantiated dynamically, the Squid service and the Shoal Agent are installed on the startup of the machine, using Puppet apply. The shell script to execute the Puppet apply command is included as part of the launch of the virtual machine and uses the cloud-init features of AWS, or the contextualization features of OpenNebula respectively. We used the existing Puppet Forge modules for Squid as part of this work.

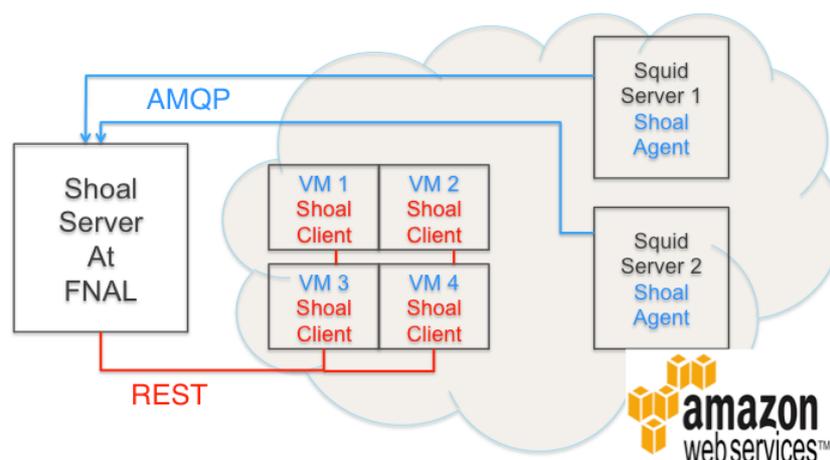


Figure 1: Architecture of Dynamically Instantiated EC2 Instances with On-Demand Caching Service and a Cache Publishing Service

When a worker node is instantiated, the Shoal Client, the CVMFS client, and other unrelated software are installed at the startup of the machine via a startup script that uses Puppet apply. This is

done on FermiCloud. We use publically available Puppet modules for the CVMFS client. It could be done on AWS as well but in practice we pre-install it and send it along with our virtual machine worker node image. The Shoal Client is a cron job that queries the Shoal Server using the REST interface to get the closest Squid Server and is configured to run every 2 hours. Shoal Client updates the proxy address in the CVMFS configuration file. When CVMFS client tries to download any software from CVMFS server, the request passes through the configured Squid Server. The Shoal Client was later modified to modify the system configuration files to also send other non-CVMFS related traffic such as fetching certificate revocation lists to the dynamically detected nearest squid server. Experimental software may also need to be reconfigured to use the dynamically detected nearest squid server.

Our goal was to have the FermiCloud VMs on the private net use the on-site Squid servers and the AWS VMs use Squid servers that were instantiated on AWS. In this way we did not have to open the Fermilab Squid servers to the outside Internet and we also delivered a much faster Squid service due to decreased latency. We found that one m3.large Squid server in AWS (2 cores, 7.5GB RAM, 30 GB Disk) was sufficient to serve the load for caching code for 1000 running jobs. Any element code would only be fetched once to the AWS squid server and all other AWS virtual machines would access it from there.

### *2.3. Virtual machine conversion system*

We also required a faster way to upload specialized virtual machines to Amazon Web Services. Our goal was to run on AWS an image as close as possible to the one used on private deployments. We leveraged our existing mechanism that manufactures our stock image for the private cloud and added extra steps to it that strip out a few Fermilab-specific configurations and add a few Amazon-specific configurations. The standard image is uncompressed from qcow2 to raw format. Then it is copied to another running image on AWS which has an extra disk mounted. The extra disk is then saved as a snapshot. Once the virtual machine image is stored on Amazon then all copies of the virtual machines are launched from that. For our worker node virtual machines we used the Amazon m3.large instance type as mentioned above because it had a good match of available cores (2) and enough scratch space (30GB SSD) to host two jobs at once.

### *2.4. Private cloud scalability improvements*

Our private cloud, FermiCloud, was running OpenNebula 3.2 at the time. The “econe” emulation of the AWS EC2 API was functional for launching small numbers of virtual machines and had been used at times when an experiment needed a batch slot with memory larger than the then-default 2GB per batch slot on FermiGrid. There were known problems if you tried to launch 20 or more virtual machines. The file system of GFS2 on SAN was adequate for the initial 23 hosts that it served but could not be expanded to a large scale. In addition we did not have public IP address space readily available to launch 1000 more virtual machines on the public network.

140 old Dell PowerEdge 1950 machines with dual quad-core Xeon processors and 16 GB of RAM were made available for this test. A private virtual LAN was made available to these nodes for use by the virtual machines. This private LAN was assigned a routable private network which could be reached from anywhere inside Fermilab. We installed a new test head node based on OpenNebula 4.8. For an image repository we used space on our BlueArc NAS server. This proved to be quite scalable.

### *2.5. Job Submission and Provisioning*

The NOVA experiment users used the new client/server “Jobsub” submission system to submit their jobs. This is the standard job submission software that Intensity Frontier users at Fermilab use to submit to local Fermilab resource, opportunistic use on the Open Science Grid, and the private and commercial clouds. The GlideinWMS [1] provisioning system then identifies user jobs in the queue that are suitable for running on our private cloud or on commercial clouds, and starts virtual machines on the appropriate cloud if necessary by use of the EC2 Query API. Once the virtual machine starts up, there is a process at boot time that checks the integrity of the virtual machine and starts up a HTCondor daemon. This then calls back to the batch system to declare itself available to run a job. When no further jobs are available, the HTCondor daemon exits and the virtual machine then terminates itself.

GlideinWMS is well suited to distributing workloads evenly across a number of relatively homogeneous computing platforms such as local clusters, grids, and clouds. Since running on Amazon Web Services also requires awareness of available budget, we have identified a program of work to add an external policy engine to handle the financial decision making.

## *2.6. Performance Evaluation*

Figures 1a and 1b show the results of the final and largest trial, in which 1000 jobs ran simultaneously both on our local private cloud and on Amazon Web Services. The completion time of the jobs was relatively the same. The ramp-up factor on AWS was limited only by the submission rate that was configured into our GlideinWMS factory. The slower ramp-up time for jobs on our local cloud was more due to the virtual machine launching pattern we were using at the time, which caused all 8 of the virtual machines to launch on the same node simultaneously, significantly stressing the local disk. We have since switched to a scheduling algorithm that distributes the launch more equally across the cloud.

We ran a total of 3300 jobs on AWS in the largest trial, shown in Figure 1A. Each job generated on average 250MB of output, the total output was 467GB for which we incurred \$51 in data transfer charges. We incurred \$398 in virtual machine charges, with a peak of 525 virtual machines running. This work was done with “On-demand” instances. It could have been done much more cheaply with “spot pricing” from Amazon and our current program of work is now doing this on a regular basis. We also evaluated the possibility of storing the data temporarily to the Amazon Simple Storage Service (S3) before staging it back. For this use case with a fairly small number of machines and small data, it proved to be both faster and cheaper just to stage directly back to Fermilab. We expect that we will at some point need to do staging for more data-intensive tasks and have developed S3-aware data movement tools for this purpose.

## **3. On-demand scalable services**

The summer of 2014 testing demonstrated the successful operation of a classical service discovery model of locating a Squid server. Since then we have demonstrated an automated launch of a Squid service in Amazon Web Services. This service makes use of the Elastic Load Balancer service to provide a single entry point to the service, which can have multiple Squid servers backing it up, and an Auto Scaling Group to automatically add more Squid servers when the load goes higher, and remove them when the load goes down. A CloudFormation service can be used to start the whole stack of services. We plan to use native cloud scaling mechanisms like this wherever possible in future as we address cloud workflows of 100-1000 times larger than the ones described in this work. Amazon’s Route53 service is used to attach a predictable address to the scalable Squid service.

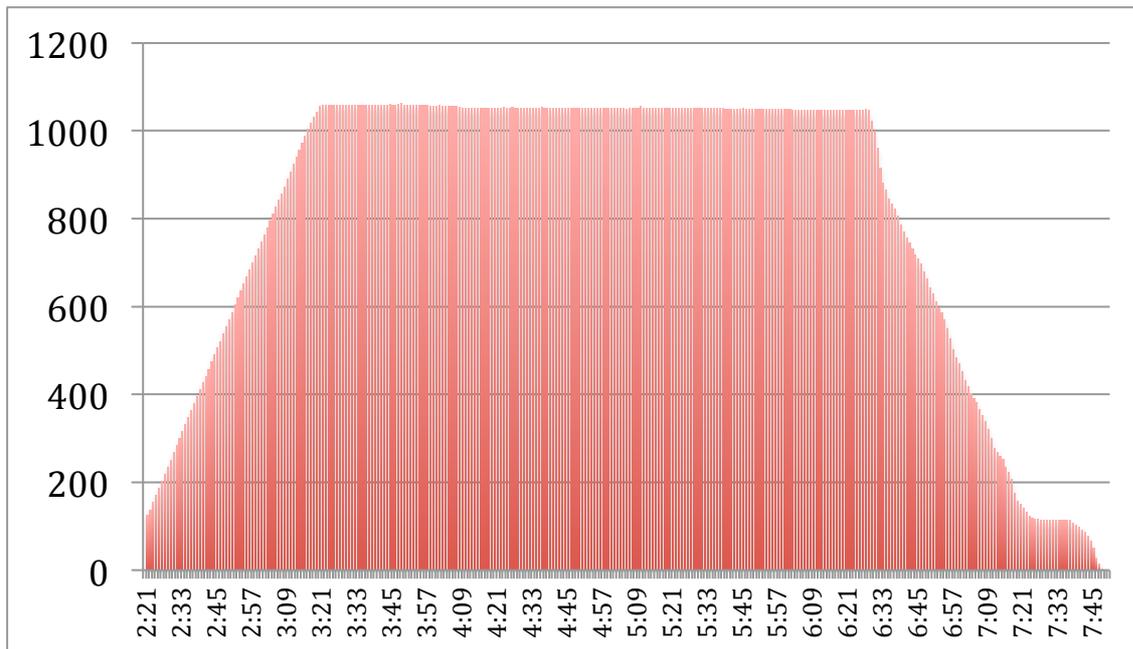


Figure 2a. Number of jobs running as a function of time of day, Amazon AWS

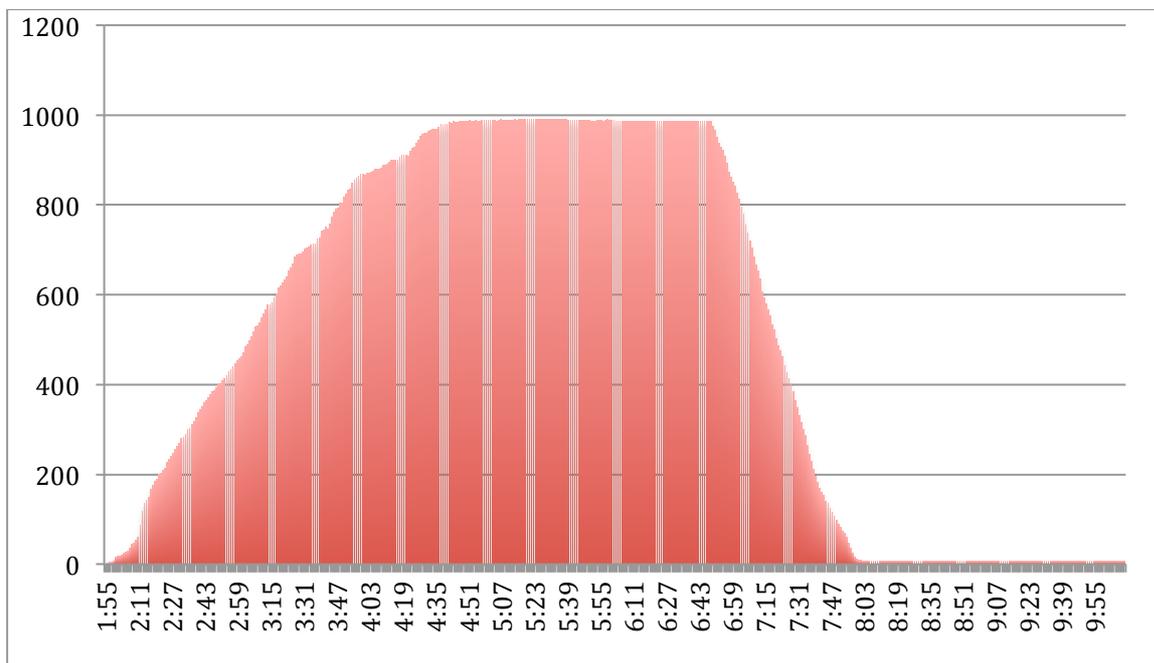


Figure 2b. Number of jobs running as a function of time of day, Fermilab private cloud

#### 4. Conclusions and Future Work

We have demonstrated the capacity to successfully run an Intensity Frontier application at scale both in the public and the private cloud. The cloud submission capacities remain available to our production users. Our future program of work will now focus on integrating the commercial cloud

providers more closely into our facility operations. The overall goal is to make the presence of the cloud resources be transparent to our end users, running even the most data-intensive applications on the cloud if necessary. We are partnering with the US research networks and commercial cloud providers to facilitate this work. We are grateful especially for all the work that was done by the NOvA experimenters to help make this happen.

### **Acknowledgements**

Work supported by the U.S. Department of Energy under contract No. DE-AC02-07CH11359, and by CRADA FRA 2014-0002 / KISTI-C14014.

### **References**

- [1] Sfiligoi, I., Bradley, D. C., Holzman, B., Mhashilkar, P., Padhi, S. and Wurthwein, F. (2009). "The Pilot Way to Grid Resources Using glideinWMS", 2009 WRI World Congress on Computer Science and Information Engineering, Vol. 2, pp. 428–432. [doi:10.1109/CSIE.2009.950](https://doi.org/10.1109/CSIE.2009.950).
- [2] J. Blomer et al, Status and future perspectives of CernVM-FS J. Phys.: Conf. Ser. 396052013, [doi:10.1088/1742-6596/396/5/052013](https://doi.org/10.1088/1742-6596/396/5/052013)
- [3] Squid - HTTP proxy server <http://www.squid-cache.org>, 2015
- [4] Gable, Ian, et al. Dynamic web cache publishing for IaaS clouds using Shoal. *Journal of Physics: Conference Series*. Vol. 513. No. 3. IOP Publishing, 2014.
- [5] I. Gable et al, A batch system for HEP applications on a distributed IaaS cloud J. Phys.: Conf. Ser. 331062010, [doi:10.1088/1742-6596/331/6/062010](https://doi.org/10.1088/1742-6596/331/6/062010)
- [6] Python Package Index <https://pypi.python.org/>, 2015
- [7] RabbitMQ - AMQP Messaging software, <http://www.rabbitmq.com>, 2015
- [8] S.Vinoski, Advanced Message Queuing Protocol, IEEE Internet Computing 1087, [doi:10.1109/MIC.2006.116](https://doi.org/10.1109/MIC.2006.116)
- [9] A. Norman, The NOvA Experiment, A Long-baseline Experiment at the Intensity Frontier. PoS, HQL2012 (2012).

# Virtual Facility at Fermilab: Infrastructure and Services Expand to Public Clouds

---

Steven Timm<sup>1</sup>

Fermilab

*P.O. Box 500, Batavia, IL 60510, USA*

E-mail: [timmm@fnal.gov](mailto:timmm@fnal.gov)

Gabriele Garzoglio

Fermilab

*P.O. Box 500, Batavia, IL 60510, USA*

E-mail: [garzogli@fnal.gov](mailto:garzogli@fnal.gov)

Stuart Fuess

Fermilab

*P.O. Box 500, Batavia, IL 60510, USA*

E-mail: [fuess@fnal.gov](mailto:fuess@fnal.gov)

Glenn Cooper

Fermilab

*P.O. Box 500, Batavia, IL 60510, USA*

E-mail: [gcooper@fnal.gov](mailto:gcooper@fnal.gov)

## Abstract:

In preparation for its new Virtual Facility Project, Fermilab has launched a program of work to determine the requirements for running a computation facility on-site, in public clouds, or a combination of both. This program builds on the work we have done to successfully run experimental workflows of 1000-VM scale both on an on-site private cloud and on Amazon AWS. To do this at scale we deployed dynamically launched and discovered caching services on the cloud. We are now testing the deployment of more complicated services on Amazon AWS using native load balancing and auto scaling features they provide.

The Virtual Facility Project will design and develop a facility including infrastructure and services that can live on the site of Fermilab, off-site, or a combination of both. We expect to need this capacity to meet the peak computing requirements in the future. The Virtual Facility is intended to provision resources on the public cloud on behalf of the facility as a whole instead

---

<sup>1</sup>

Speaker

of having each experiment or Virtual Organization do it on their own. We will describe the policy aspects of a distributed Virtual Facility, the requirements, and plans to make a detailed comparison of the relative cost of the public and private clouds. This talk will present the details of the technical mechanisms we have developed to date, and the plans currently taking shape for a Virtual Facility at Fermilab.

The International Symposium on Grids and Clouds (ISGC) 2015  
*March 15-20, 2015*  
Academia Sinica, Taipei, Taiwan

POS ( ISGC2015 ) 014

## 1. Introduction

The Fermilab scientific program includes several running experiments, both the CMS experiment at the Energy Frontier, and the various neutrino and muon experiments on the Intensity Frontier. The ongoing data analysis and simulation for running experiments, combined with a large simulation load for future facilities and experiments, results in an unprecedented level of computing demand. This paper describes recent progress in the ongoing program of work to expand our computing to the distributed resources of grids and public clouds. We will also describe our plans for the Virtual Facility Project, which will integrate these technologies into our computing facility.

### 1.1 Definitions

In this paper the Facility refers to all computing resources and storage resources that are provided on behalf of the users. This includes on-site resources, friendly grid sites, opportunistic grid usage, and use of private, community, and commercial clouds. The Virtual Facility consists of the resources that we dynamically provision, plus the services that allow us to provision virtual machines and grid slots, as well as the auxiliary services that are required to make analysis possible at remote sites such as the public cloud. Provisioning is defined as the process of contacting grid and cloud sites on behalf of the users to obtain batch slots for job execution.

### 1.2. Background

The process of provisioning job slots has historically been done by individual Virtual Organizations. At Fermilab the GlideinWMS system has been used for this purpose. It is designed to request resources from a pre-configured set of hosts, both local and remote. These resources are presented as a unified virtual resource pool to the users, commonly referred to as the “user pool”. The combination of unified provisioning and automated credential management has proven a powerful combination to get thousands of users doing grid computation on a regular basis. The developers of GlideinWMS and the HTCondor system on which it relies have worked with various cloud providers over the years to support submission of virtual machines to various public and private clouds including Amazon Web Services, Google Compute Engine, OpenNebula, and OpenStack. The virtual machines submitted by GlideinWMS start up a condor daemon, which calls back to the user pool and is ready to accept a job.

### 1.3 Business Case for Commercial Clouds

The workloads of our current experiments are stochastic in nature. There is some amount of continuous usage, punctuated by large peaks of usage. This is often related to extra analysis that must be done just before conferences. The peaks of demand can be up to four times the regular usage. We believe it is not financially feasible to size our computing clusters to handle the absolute peak demand. Nor can we rely on opportunistic grid slots to fill the gap because when our site is under high pre-conference demand the rest of them are likely to be

busy as well. It is therefore worthwhile to carefully consider commercial clouds as an option to meet this peak demand. We know it is technically possible to run on the commercial cloud. What we need to demonstrate now is sustainability and efficiency, as well as financial feasibility.

## 2. Workflows on cloud to date

As part of the joint collaboration between Fermilab and KISTI, we have a program of work building towards distributed federated clouds. In the summer of 2014 the primary goal of this program was to demonstrate a federated cloud running at the 1000 Virtual Machine scale, using our local private cloud nodes and Amazon Web Services EC2. Our application of choice for this is the Cosmic Ray simulation of the NOvA experiment far detector at Ash River. This application requires negligible input and produces about 250MB of output per job, and is quite computationally intensive. The NOvA experimenters spent considerable effort in optimizing their code to run at various sites outside of Fermilab, including loading their code into the OSG OASIS CVMFS server. The NOvA experiment supplied us with a set of files and scripts, which would run one full set of their cosmic ray Monte Carlo, 20000 input files in all, with one job per file.

### 2.1 Scaling OpenNebula 4 cloud to 1000 Virtual Machines

OpenNebula cloud supports launching virtual machines via its emulation of the EC2 interface. The OpenNebula 3 cloud in production since 2012 supported this function but an OpenNebula upgrade to version 4.8 was necessary to support the bulk launching at the 1000 Virtual Machine level. We also added 140 worker nodes for the purpose of this test. These were 8-core Dell PowerEdge 1950 servers that were formerly part of the CDF clusters at Fermilab. The worker nodes were attached to a routable private network, which could access all network points inside of Fermilab but not outside. We used the Bluearc NAS server as an image store to distribute the operating image, which is stored in the compressed “qcow2” format. A puppet script applied at boot time did configuration management on the image. This was used to install the CVMFS client software and a few other prerequisites including certificate authority files. Since the goal was to test 1000 virtual machines, we launched 1000 virtual machines with one core apiece. Under normal operation we would run virtual machines with four cores apiece.

### 2.2 Scaling Amazon Web Services to 1000 Virtual Machines

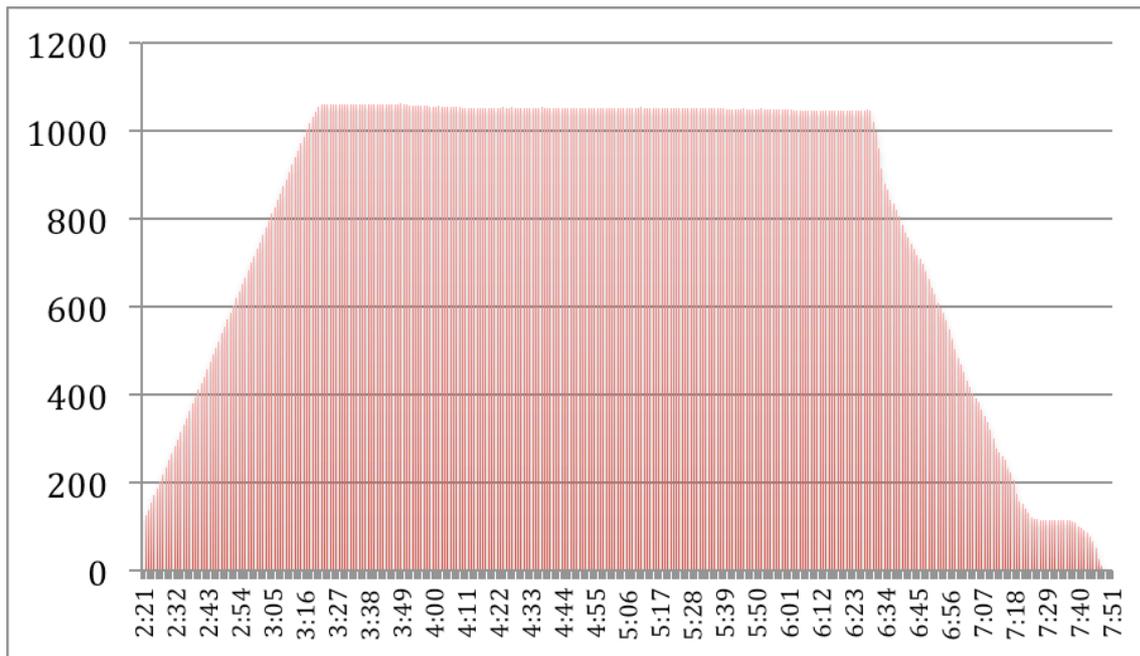
Earlier smaller trials of running jobs on AWS had shown us that the bottleneck was the proxy caching of the code on AWS. We had been using a Squid proxy server at Fermilab to cache the CVMFS code, which is not optimal for security reasons or for bandwidth reasons. So we found it necessary to launch one or more squid services in AWS itself. Since these services are dynamically instantiated there is no a priori way to know what the IP address of the squid server will be at any given time. We used the SHOAL discovery system that was written for this purpose by the team at the University of Victoria. In this system, each squid server launched in the cloud runs a Shoal Agent that contacts a fixed server at Fermilab via the AMQP messaging protocol. Each worker node VM then runs a client which calls the Shoal server at Fermilab which returns a list of possible squid servers, sorted by order of which is closest to the local VM network-wise. In practice we found that a single squid server in the cloud was sufficient to serve 500 virtual machines.

We also required a faster way to upload specialized virtual machines to Amazon Web Services. It was our goal to run a very similar virtual machine image on Amazon as we run on our private cloud, based on Scientific Linux. We leveraged our existing mechanism that manufactures our stock image for the private cloud and added extra steps to it that strip out a few Fermilab-specific configurations and add a few Amazon-specific configurations. Then it is copied to another running image on AWS which has an extra disk mounted. The extra disk is then saved as a snapshot. Once the virtual machine image is stored on Amazon then all copies of the virtual machines are launched from that. We settled on the Amazon m3.large instance type because it had two available cores and enough default scratch space (30GB SSD) to host two jobs at once.

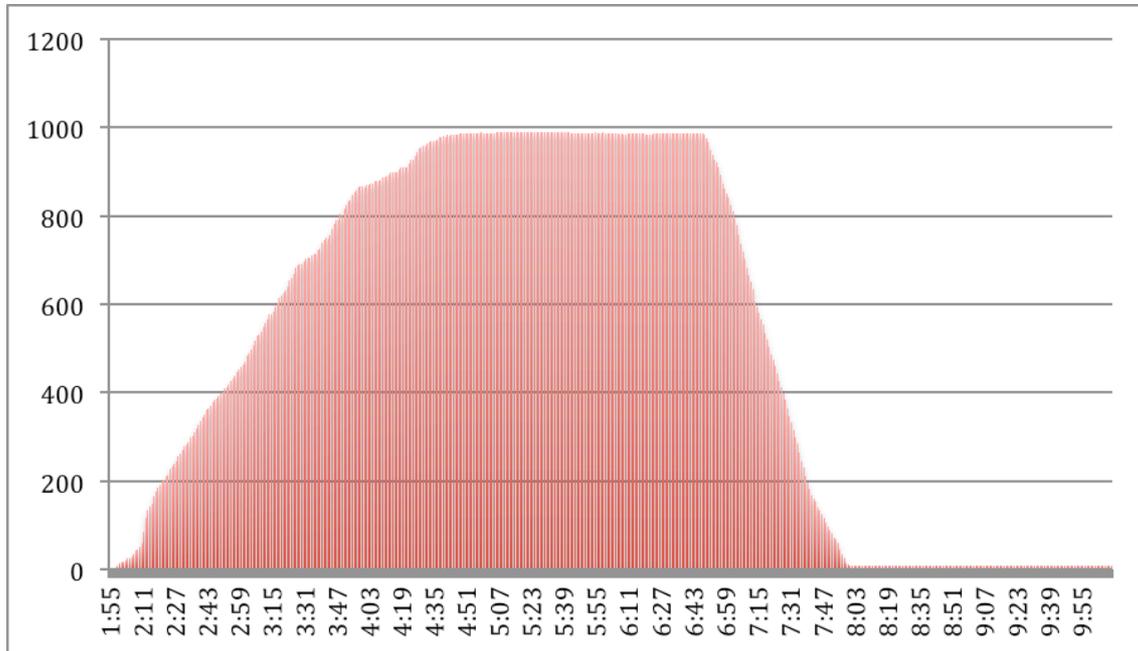
### 2.3 Results

Figures 1a and 1b show the results of the final and largest trial, in which 1000 jobs ran simultaneously both on our local private cloud and on Amazon Web Services. The completion time of the jobs was relatively the same. The ramp-up time for jobs on our local cloud was more due to the virtual machine launching pattern we were using at the time, which caused all 8 of the virtual machines to launch on the same node simultaneously, significantly stressing the local disk. We have since switched to a scheduling algorithm, which distributes the launch more equally across the cloud.

We ran a total of 3300 jobs on AWS in the largest trial, shown in Figure 1A. Each job generated on average 250MB of output, the total output was 467GB for which we incurred \$51 in data transfer charges. We incurred \$398 in virtual machine charges, with a peak of 525 virtual machines running.



POS (ISGC2015) 014

**Figure 1a. Number of jobs running as a function of time of day, Amazon AWS****Figure 1b. Number of jobs running as a function of time of day, Fermilab private cloud**

### 3. Virtual Facility—Expanding the facility to the public cloud

It is now the goal to integrate the commercial cloud running of batch jobs into the Fermilab Facility on a transparent basis. We currently have a unified job submission service based on the “Jobsub” client/server system. Users can currently specify usage model of private cloud or paid cloud to run on the private cloud or the paid cloud respectively. It is our goal to shift this paradigm so that the user will specify flags that specify whether their job can run off-site or not, and then the facility decides transparently whether or not to send it to the commercial cloud. We also plan to incorporate the provisioning functions of GlideinWMS into the facility. In this way we will be able to request resources on behalf of the whole facility with a single set of credentials, rather than requesting them on behalf of each of the approximately 20 active experiments we support. We describe below some of the major work items that will be necessary to do in this process.

#### 3.1 Data intensive production

Intensity Frontier data production is typically more data intensive than the cosmic ray example that we used in the summer 2014 demonstration. For the commercial clouds to be a transparent extension of our computing facility, we will have to successfully and regularly do data-intensive data reconstruction and Monte Carlo simulation on the public cloud. For

example, a typical data reconstruction campaign for the NOvA experiment is expected to have 10000 input files on average, 250MB in size, and produce one 250MB output file for each, taking 3 hours per job. A typical Monte Carlo simulation campaign of a neutrino beam could take 38000 input files of average 250MB in size and make 250MB of output, taking up to 5 hours per job. We have proposed a program of running six data reconstruction campaigns and ten Monte Carlo simulation campaigns on Amazon over the course of the next year. This will be approximately 2.1 million hours of compute time overall. For scale, the NOvA experiment used approximately 10.2 million hours of compute time total in 2014, and Fermi Grid as a whole has capacity for 145 million CPU-hours during the course of a calendar year.

Given the increased amount of data that will be produced by these jobs, we believe it will be necessary to do local caching of the output results on the S3 service of Amazon before bringing the data back to Fermilab. In this way we do not have hundreds of virtual machines waiting for 30-45 minutes just to transfer their data back to Fermilab. There will likely be some modifications necessary to our higher-level data handling software to support this.

### 3.2 Provisioning Algorithms

Currently GlideinWMS operates on a demand-based system. If there are user jobs which request a particular resource type such as off-site cloud, then GlideinWMS will continue trying to contact the off-site resource and to gain more of those resources, until there are a fixed number of idle jobs waiting and it is apparent that the resource is full. Since commercial clouds have very large capacity and cost us for each machine we start, a different model is required.

We have proposed a call-out architecture to enable the GlideinWMS Frontend to call out to an external routine to determine whether it should request more glideins from the public cloud or not. The goal of this routine will be to optimize the job placement based on expected execution time as well as overall cost. In general if the local resources are full, and expected to remain full for the length of time that it takes to launch virtual machines on AWS, then we would launch VM's on AWS. We may, however, wish to restrict AWS only to certain users.

A number of other small improvements will be needed to the GlideinWMS system. These include better support for AWS spot pricing, capacity to submit simultaneously to multiple AWS regions and availability zones, selection of various resource types and AMI id's.

### 3.3 On-Demand Services in the Commercial Cloud

A number of auxiliary services may eventually be necessary in the public cloud. We certainly need Squid for code caching via CVMFS. We may well need to have a temporary storage element on the public cloud to aid in data caching inbound and outbound. We might need to set up a CVMFS Stratum 1 server or a node that serves Alien Cache (an extension to CVMFS that allows for serving large files that are not c

We demonstrated the Squid service operation in the cloud previously, but it had to be manually launched. It would be preferable to have a known service address that can be

activated dynamically when it is needed and scale to the size that is needed. This functionality can be done on Amazon Web Services using a combination of the Elastic Load Balancer, and the Autoscaling group. The Autoscaling group requires one server to be up at all times, however, so to truly launch the service on demand only as needed, an orchestration script such as CloudFormation is necessary to launch the whole group of services. Squid servers are basically stateless and thus are good candidates for scaling up and down.

It is more challenging to scale a stateful service such as a job submission service up and down, since these servers have locally stored log files from the jobs that have to be submitted, that need to be accessible for some length of time. We demonstrated a scalable job submission service on AWS by means of lifecycle hooks and the new “standby” state. Finally, it is possible to create a failover alias that will attempt to contact the public cloud service first and fall back to a service on site if it is not there. By these means it is possible to define a service set that spans both public cloud and local site.

Many if not most of the services mentioned on Amazon Web Services have counterparts in open source cloud software such as OpenStack and OpenNebula as well as other public cloud providers. Any mention of Amazon Web Services in this paper should not be interpreted to exclude future collaboration with other commercial cloud providers.

Following work that has been done at Brookhaven National Labs, we will also configure the virtual machines on Amazon and the network routers on both ends to send network traffic over the ESNET link.

#### **4. Conclusion**

The Virtual Facility Project is using many of the long-standing tools of distributed computing such as virtual machines, HTCondor, GlideinWMS, and Amazon Web Services. What we bring is a new emphasis. We will ask the hard policy and architecture questions needed to integrate the commercial cloud into the Facility. We will make full use of cloud features and use the cloud like a cloud, not as an extension of the grid. We will explore strategic partnerships with the commercial cloud vendors. We will do the hardest data-intensive computing on the public cloud. We will unify grid and cloud provisioning activities into the Facility, saving both system overhead and personnel for several Virtual Organizations.

#### **Acknowledgements**

This work is supported by the US Department of Energy under contract number DE-AC02-07CH11359 and by KISTI under a joint Cooperative Research and Development Agreement. CRADA-FRA 2014-0002/ KISTI-C14014

## References

- [1] J. Blomer et al, Status and future perspectives of CernVM-FS J. Phys.: Conf. Ser. 396052013, doi:10.1088/1742-6596/396/5/052013
- [2] I. Gable et al, A batch system for HEP applications on a distributed IaaS cloud J. Phys.: Conf. Ser. 331062010, doi:10.1088/1742-6596/331/6/062010
- [3] Gable, Ian, et al. "Dynamic web cache publishing for IaaS clouds using Shoal." *Journal of Physics: Conference Series*. Vol. 513. No. 3. IOP Publishing, 2014.
- [4] Wu, Hao, et al. "Automatic cloud bursting under fermicloud." *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*. IEEE, 2013.
- [5] Timm, S., et al. "Grids, virtualization, and clouds at Fermilab." *Journal of Physics: Conference Series*. Vol. 513. No. 3. IOP Publishing, 2014.

```

1 | File: predictiondecisionengine/trunk/bin/aws_launch_benchmark.sh
2 | #!/bin/bash
3 | . /home/crivella/bin/functions.sh
4 |
5 | #Check if needed packages are installed
6 | if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
7 |     echo "This script require aws-cli to run."
8 |     echo "Follow the instructions on https://aws.amazon.com/cli/ on how to get it"
9 |     exit
10 |
11 | fi
12 | if [[ `which nmap` = "" || `which nmap | grep -F "no nmap" != "" ]]; then
13 |     echo "This script require nmap to work. Install it by writing sudo apt-get install nmap"
14 |     echo "...Or the equivlaent command for your distribution"
15 |     exit
16 | fi
17 |
18 | re='^[0-9]+$'
19 |
20 | #Edit this part with your data
21 | #Where you have the run-*.sh files
22 | FILES_PATH=$HOME/bin/Fermilab
23 | #Where you stored your .pem keys
24 | KEYS_PATH=$HOME/Fermilab/Keys
25 | #where to keep the record of the machines you launched
26 | TMP_PATH=$HOME/Fermilab/tmp
27 |
28 | #Your username
29 | USER=grassano
30 | #ID of the AMI you will be using
31 | AMI_ID=ami-03dec833
32 | #Just the name with no .pem, or modify rest of the script
33 | KEY=usw_oregon_grassano
34 | #ID of the security group
35 | SG_ID=sg-91d414f5
36 | #Not the same for every instance
37 | SAME_INSTANCE_LIMIT=25
38 | #Bucket to get the benchmark files from

```

```

39 | TEST_BUCKET=grassano-test
40 |
41 | #Benchmark that the script can run
42 | declare -a benchmarks=(tt_bar_gensim tt_bar_reco hepspec06 s3_stresstest_d s3_stresstest_u
... test_fermigrd test_fermigrd2)
43 | #Volume to add with required size for the benchmark
44 | declare -a req_vol_size=(80 80 50 0 3 3 3)
45 | #Role
46 | declare -a roles=(""" "" " "" " AllowS3_Upload """)
47 | declare -a parameters=(""" "" " "1 10 100" "1 10 100" "1 5 10 20" "1 5 10 20")
48 |
49 | #Check for registered user
50 | USER_DATA=`aws iam get-user`
51 | USER_ARN=`echo "$USER_DATA" | grep Arn | cut -d "\"" -f 4`
52 | if [[ -z "$USER_ARN" ]]; then
53 |     echo "You don't appear to be registered to aws services. Exiting"
54 | else
55 |     USER=`echo "$USER_DATA" | grep UserName | cut -d "\"" -f 4`
56 |     USER_ID=`echo "$USER_ARN" | cut -d ":" -f 5`
57 |     #echo $USER:$USER_ID
58 | fi
59 |
60 | declare -a instances=(1:t2.micro)
61 | RESUME_STATE="false"
62 | ZONE=""
63 | while getopts hra:s:k:i:p:z: opt; do
64 |     case $opt in
65 |         a) AMI_ID=$OPTARG
66 |             ;;
67 |         s) SG_ID=$OPTARG
68 |             ;;
69 |         k) KEY=$OPTARG
70 |             ;;
71 |         i) unset instances
72 |             C_COUNT=0
73 |             for CMD in $OPTARG; do
74 |                 if [[ ! `echo $CMD | cut -d ":" -f 1` =~ $re || `echo $CMD | grep ":" -o | wc -l` -gt 1
... ]]; then

```

```

75     echo "Invalid parameter for -i. Use the -h formore info."
76     exit
77 fi
78 instances[$C_COUNT]=$CMD
79 let C_COUNT++
80 done
81
82 ;;
83 p) PARAM=$OPTARG
84 ;;
85 z) if [[ `grep "profile $OPTARG" $HOME/.aws/config` != "" ]]; then
86     ZONE="--profile $OPTARG"
87 else
88     echo "You don't have a profile to work with this zone. Exiting..."
89     exit
90 fi
91 ;;
92 r) RESUME_STATE="true"
93 if [[ "$OPTARG" != "" ]]; then
94     echo "The -r does not take additional parameter. Use the -h for additional info"
95     exit
96 fi
97 ;;
98 h | \?) cat<<EOU
99
100 Usage: $0 [-a AMI_ID] [-s Security_group_ID] [-k KEY_NAME] [-i instances] [-r] [-p parameters]
... [z zone]
101 -a AMI_ID          ID of the AMI to use
102 -s Security_group_ID  ID of the security group to use
103 -k KEY_NAME        Specify the name of the key to use without .pem
104 -i instances       List of instances to start es: "3:m3.medium 2:c4.2xlarge"
105 -p parameters      Parameters to pass to the run-*.sh script as "... .. ."
106 -z zone            Region to launch the instances in
107 -r                Enable resume mode
108 EOU
109     exit
110     ;;
111 esac

```

```

112 done
113
114 #Check if the selected AMI is available, if not give option to chose from list of owned
... availabe AMIs
115 while [[ `aws ec2 describe-images $ZONE --image-id $AMI_ID 2>&1 | grep "\"Name\":" = ""` ]];
... do
116     echo "The specified ami does not exist. Do you wish to choose one from the ones you
... own?(y/n)"
117     read_choice 5 "y" "n" "yes" "no"
118     if [[ `echo $CHOICE | grep y` = "" ]]; then
119         exit
120     else
121         IMAGES=`aws ec2 describe-images $ZONE --filters Name=state,Values=available --owners
... $USER_ID`
122         IMAGES_NAME=`echo "$IMAGES" | grep "\"Name\":" | cut -d "\"" -f 4`
123         IMAGES_ID=`echo "$IMAGES" | grep "\"ImageId\":" | cut -d "\"" -f 4`
124         make_list "$IMAGES_NAME"
125         read_choice 5 `seq $LENGHT`
126         AMI_ID=`echo "$IMAGES_ID" | head -n $CHOICE | tail -n 1`
127         #echo $AMI_ID
128     fi
129     unset IMAGES
130     unset IMAGES_NAME
131     unset IMAGES_ID
132 done
133
134 #Check if the chosen key is available and recognized by aws
135 while [[ `find $KEYS_PATH -name "$KEY.pem" = ""` || `aws ec2 describe-key-pairs $ZONE | grep
... $KEY` = "" ]]; do
136     echo "You dont own the specified key or it is not recognized by amazon."
137     echo "Do you wish to choose it from the ones listed in aws that you own as well?(y/n)"
138     read_choice 5 "y" "n" "yes" "no"
139     if [[ `echo $CHOICE | grep y` = "" ]]; then
140         exit
141     else
142         KEYS=`find $KEYS_PATH -name "*.pem"`
143         KEYS_DATA=`aws ec2 describe-key-pairs $ZONE`
144         C_COUNT=0

```

```

145     while read -r line; do
146         line=${line::-4}
147         FIELD=`echo $line | grep "/" -o | wc -l`
148         let FIELD++
149         KEY=`echo $line | cut -d "/" -f $FIELD`
150         if [[ `echo "$KEYS_DATA" | grep "$KEY" != ""` ]]; then
151             KEY_LIST[$C_COUNT]="$KEY"
152             let C_COUNT++
153         fi
154     done <<< "$KEYS"
155     if [[ $C_COUNT -lt 1 ]]; then
156         echo "No matching key found. Exiting..."
157         exit
158     fi
159     make_list "${KEY_LIST[@]}"
160     read_choice 5 `seq $LENGHT`
161     let CHOICE--
162     KEY="${KEY_LIST[$CHOICE]}"
163     #echo $KEY
164 fi
165 unset KEY_LIST
166 unset KEYS
167 unset KEYS_DATA
168 done
169
170 ##Check if the selected SG is available, if not give option to chose from list of availabe SGs
171 while [[ `aws ec2 describe-security-groups $ZONE --group-ids $SG_ID 2>&1 | grep
... "\GroupId\":" = ""` ]]; do
172     echo "The specified security group does not exist.Do you wish to chose one from those
... available on aws?(y/n)"
173     read_choice 5 "y" "n" "yes" "no"
174     if [[ `echo $CHOICE | grep y` = ""` ]]; then
175         exit
176     else
177         SGS=`aws ec2 describe-security-groups $ZONE`
178         SGS_NAME=`echo "$SGS" | grep "\GroupName\":" | cut -d "\"" -f 4`
179         #echo "$SGS_NAME"
180         SGS_ID=`echo "$SGS" | grep "\GroupId\":" | cut -d "\"" -f 4`

```

```

181     make_list "$SGS_NAME"
182     read_choice 5 `seq $LENGHT`
183     SG_ID=`echo "$SGS_ID" | head -n $CHOICE | tail -n 1`
184     #echo $SG_ID
185 fi
186 unset SGS
187 unset SGS_NAME
188 unset SGS_ID
189 done
190
191 mkdir -p $TMP_PATH/launch_logs
192
193 echo "Chose the kind of benchmarks to execute:"
194
195 make_list "${benchmarks[@]}"
196 read_choice 5 `seq $LENGHT`
197 let CHOICE--
198
199 benchmark=${benchmarks[$CHOICE]}
200 vol_size=${req_vol_size[$CHOICE]}
201
202 if [[ -z "$PARAM" ]]; then
203     PARAM=${parameters[$CHOICE]}
204 fi
205
206 if [[ $vol_size -eq 0 ]]; then
207     vol_size=3
208     for CMD in ${PARAM[@]}; do
209         let vol_size+=$CMD
210     done
211 fi
212
213 #If a role is associated with the benchmark, enable it for the run-instances command
214 if [[ "${roles[$CHOICE]}" = ""` ]]; then
215     ROLE="AllowS3_Download"
216 else
217     ROLE="${roles[$CHOICE]}"
218 fi

```

```

219
220 ROLE="--iam-instance-profile Name=\"\$ROLE\""
221
222 #echo "$benchmark:$vol_size:$ROLE"
223 #echo "${instances[@]}"
224
225 #####
... #####
226 #Check if the TEST_BUCKET in s3 contains a folder
227 #This script wont work if all the require files aren't in an s3 bucket in a folder with the
... Name set as the benchmark name
228 #Exception for s3_stresstest_d/u and test_fermigrid that go with the same foder s3_stresstest
229
230 if [[ `echo $benchmark | grep s3_stresstest` != "" || `echo "$benchmark" | grep
... "test_fermigrid" != "" ]]; then
231     a_benchmark="s3_stresstest"
232 else
233     a_benchmark=$benchmark
234 fi
235
236 if [ "`aws s3api list-objects --bucket $TEST_BUCKET | grep $a_benchmark`" = "" ]; then
237     echo "Can't find the right folder in s3. Quitting..."
238     exit
239 fi
240
241 #echo -e "$instances\n$PARAM\n$benchmark:$vol_size"
242 #exit
243
244 #####
... #####
245 #Create folder to store the VMS data + Resume work option
246 COUNT2=0
247 RESUME=0
248 RES_STEP=0
249 mkdir -p $TMP_PATH/$benchmark
250 if $RESUME_STATE; then
251     if [[ ! ${#instances[@]} -eq 1 ]]; then
252         echo "The resume functionality works only with one kind of instances and benchmark."

```

```

253     exit
254 fi
255 if [[ ! -f $TMP_PATH/$benchmark/VMS ]]; then
256     echo "Tmp file not found. Exiting..."
257     exit
258 fi
259 while read -r line; do
260     if [[ "$line" != "" ]]; then
261         I_CHECK=`echo ${instances[0]} | cut -d ":" -f 2`
262         I_TYPES[$COUNT2]=`echo "$line" | cut -d ":" -f 2`
263         if [[ "${I_TYPES[$COUNT2]}" != "$I_CHECK" ]]; then
264             echo "The specified instance type does not match the ones in the tmp file."
... Can't use the resume functionality"
265             exit
266         fi
267         VM_IDS[$COUNT2]=`echo "$line" | cut -d ":" -f 3`
268         let COUNT2++
269         let RESUME++
270     fi
271 done < $TMP_PATH/$benchmark/VMS
272
273 for ((COUNT=0;COUNT<COUNT2;COUNT++)); do
274     INSTANCE_DATA=`aws ec2 describe-instances $ZONE --instance-ids ${VM_IDS[$COUNT]}`
275     VM_AVAL[$COUNT]=`echo "$INSTANCE_DATA" | grep -F "AvailabilityZone" | cut -d "\"" -f
... 4`
276     VM_PUB_DNS[$COUNT]=`echo "$INSTANCE_DATA" | grep PublicDnsName | cut -d "\"" -f 4 |
... uniq`
277     ssh -o "StrictHostKeyChecking no" -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]}
... "rm -f -r /scratch/*"/dev/null 2>&1 &
278     if [[ `echo "$INSTANCE_DATA" | grep running` = "" ]]; then
279         echo "One or more of the instances described in the tmp file do not exist anymore
... or are not running."
280         echo "Can't continue with the resume functionality"
281         exit
282     fi
283 done
284 unset INSTANCE_DATA
285

```

```

286     step="Volume creation
287     Start Benchmark"
288     echo "Select the step from which you want to resume the work for the machine stored in
... tmp:"
289     make_list "$step"
290     read_choice 3 `seq $LENGHT`
291     RES_STEP=$CHOICE
292     unset step
293 else
294     rm -f $TMP_PATH/$benchmark/VMS
295     touch $TMP_PATH/$benchmark/VMS
296 fi
297
298 #####
299 #Starting the required VM
300 STEP=0
301 for ((COUNT=0;COUNT<${#instances[@]};COUNT++)); do
302     I_NUM=`echo "${instances[$COUNT]}" | cut -d ":" -f 1`
303     I_TYPE=`echo "${instances[$COUNT]}" | cut -d ":" -f 2`
304
305     if [[ $RESUME -gt 0 ]]; then
306         let I_NUM-=RESUME
307         I_COUNT=$(expr $RESUME + 1)
308     else
309         I_COUNT=1
310     fi
311     if [[ $I_NUM -gt $SAME_INSTANCE_LIMIT ]]; then
312         echo "The number ($I_NUM) specified for instance $I_TYPE exceeds the limit of
... $SAME_INSTANCE_LIMIT."
313         echo "Bringing it down to the maximun allowed..."
314         I_NUM=$SAME_INSTANCE_LIMIT
315     fi
316
317     if [[ $I_NUM -gt 0 ]]; then
318         echo "Launching $I_NUM $I_TYPE instances"
319         INSTANCE_DATA=`aws ec2 run-instances $ZONE --count $I_NUM --image-id $AMI_ID
... --instance-type $I_TYPE --security-group-ids $SG_ID --key-name $KEY $ROLE`

```

```

320     fi
321
322     L_COUNT=1
323     while [[ $L_COUNT -le $I_NUM ]]; do
324         I_TYPES[$COUNT2]=$I_TYPE
325         INSTANCE_ID=`echo "$INSTANCE_DATA" | grep -F "InstanceId" | head -n $L_COUNT | tail -n
... 1 | cut -d "\"" -f 4`
326         VM_IDS[$COUNT2]=$INSTANCE_ID
327         AVAL=`echo "$INSTANCE_DATA" | grep -F "AvailabilityZone" | head -n $L_COUNT | tail -n
... 1 | cut -d "\"" -f 4`
328         VM_AVAL[$COUNT2]=$AVAL
329         aws ec2 create-tags $ZONE --resources $INSTANCE_ID --tags
... Key=Name,Value=$USER-$I_TYPE-$benchmark-`printf %03d $I_COUNT` Key=User,Value=$USER>/dev/null
330         let I_COUNT++
331         let L_COUNT++
332         let COUNT2++
333     done
334 done
335 LAST_INSTANCE=$(expr $COUNT2 - 1)
336
337 #####
338 #Get the public DNS after they are available
339 echo "Waiting for the DNS..."
340 while [[ `aws ec2 describe-instances $ZONE --instance-ids ${VM_IDS[$LAST_INSTANCE]} | grep -F
... "PublicDnsName" | cut -d "\"" -f 4 | uniq` = "" ]]; do
341     sleep 10
342 done
343 sleep 5
344
345 #echo "$RESUME:${#VM_IDS[@]}"
346
347 for ((COUNT=$RESUME;COUNT<${#VM_IDS[@]};COUNT++)); do
348     INSTANCE_DATA=`aws ec2 describe-instances $ZONE --instance-ids ${VM_IDS[$COUNT]}`
349     PUB_DNS=`echo "$INSTANCE_DATA" | grep -F "PublicDnsName" | cut -d "\"" -f 4 | uniq`
350     #Print VMS data in tmp file
351     echo "$COUNT:${I_TYPES[$COUNT]}:${VM_IDS[$COUNT]}:$PUB_DNS:$KEY" >>
... $TMP_PATH/$benchmark/VMS

```

```

352     VM_PUB_DNS[$COUNT]=$PUB_DNS
353 done
354
355 #####
356 #Create extra volumes
357 let STEP++
358 if [[ $STEP -ge $RES_STEP ]]; then
359     RES_VAL=0
360 else
361     RES_VAL=$RESUME
362     I_COUNT=$(expr $RESUME + 1)
363 fi
364
365 for ((COUNT=$RES_VAL;COUNT<${#VM_IDS[@]};COUNT++)); do
366     if [[ "${I_TYPES[$COUNT]}" = "${I_TYPES[$(expr $COUNT - 1)]}" && $COUNT -gt $RES_VAL ]];
367     then
368         let I_COUNT++
369     else
370         I_COUNT=$(expr $RES_VAL + 1)
371     fi
372     echo "Creating extradisk for instance ${I_TYPES[$COUNT]}-`printf %03d $I_COUNT`"
373     VOLUME_DATA=`aws ec2 create-volume $ZONE --size $vol_size --availability-zone
374     ${VM_AVAL[$COUNT]} --volume-type gp2`
375     VOLUME_ID[$COUNT]=`echo "$VOLUME_DATA" | grep -F "VolumeId" | cut -d "\"" -f 4`
376 done
377 #####
378 #Wait until all VM are done initialazing
379 echo "Waiting for the instances to be initialized..."
380 for ((COUNT=$RES_VAL;COUNT<${#VM_IDS[@]};COUNT++)); do
381     while [[ `nmap -p 22 ${VM_PUB_DNS[$COUNT]} | grep open` = "" ]]; do
382         sleep 15
383     done
384 done
385

```

```

386 #####
387 #Disable the self-stop service and register the connection to the DNS (won't ask yes/no
388 afterwards)
389 for ((COUNT=0;COUNT<${#VM_IDS[@]};COUNT++)); do
390     ssh -o "StrictHostKeyChecking no" -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]}
391     "service glideinwms-pilot stop">/dev/null
392     #echo "Stopped service in instance ${instances[$COUNT]}."
393 done
394 sleep 3
395 #####
396 #Waiting for volumes to create
397 echo "Waiting for volumes to create..."
398 for ((COUNT=$RES_VAL;COUNT<${#VM_IDS[@]};COUNT++)); do
399     while [[ `aws ec2 describe-volumes $ZONE --volume-ids ${VOLUME_ID[$COUNT]} | grep
400     available` = "" ]]; do
401         #echo "Waiting for disk to create..."
402         sleep 5
403     done
404     if [[ "${I_TYPES[$COUNT]}" = "${I_TYPES[$(expr $COUNT - 1)]}" && $COUNT -gt $RES_VAL ]];
405     then
406         let I_COUNT++
407     else
408         I_COUNT=$(expr $RES_VAL + 1)
409     fi
410     aws ec2 create-tags $ZONE --resources ${VOLUME_ID[$COUNT]} --tags
411     Key=Name,Value=$USER-${I_TYPES[$COUNT]}-$benchmark-`printf %03d $I_COUNT`
412     Key=user,Value=$USER>/dev/null
413 done
414 #####
415 #Attaching Volumes
416 echo "Attaching Volumes..."
417 for ((COUNT=$RES_VAL;COUNT<${#VM_IDS[@]};COUNT++)); do
418     aws ec2 attach-volume $ZONE --volume-id ${VOLUME_ID[$COUNT]} --instance-id

```

```

414.. ${VM_IDS[$COUNT]} --device /dev/sdf>/dev/null
415 done
416
417 #####
... #####
418 #Waiting for volumes to attach
419 echo "Waiting for volumes to attach and making filesystem..."
420 for ((COUNT=$RES_VAL;COUNT<${#VM_IDS[@]};COUNT++)); do
421     while [ [ `aws ec2 describe-volumes $ZONE --volume-ids ${VOLUME_ID[$COUNT]} | grep
... attaching` != "" ] ]; do
422         sleep 3
423     done
424     if [ [ "${I_TYPES[$COUNT]}" = "${I_TYPES[(expr $COUNT - 1)]}" && $COUNT -gt $RES_VAL ] ];
... then
425         let I_COUNT++
426     else
427         I_COUNT=$(expr $RES_VAL + 1)
428     fi
429     echo "Making file system ext4 for ${I_TYPES[$COUNT]}-$I_COUNT..."
430     ssh -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]} "mkfs.ext4 /dev/xvdf
... 2>/dev/null">/dev/null &
431 done
432 wait
433
434 #####
... #####
435 #Start the benchmark
436 let STEP++
437 if [ [ $STEP -ge $RES_STEP ] ]; then
438     RES_VAL=0
439 else
440     RES_VAL=$RESUME
441 fi
442 echo "Starting the benchmark on every instance..."
443 TIME=`date +%R`
444 for ((COUNT=$RES_VAL;COUNT<${#VM_IDS[@]};COUNT++)); do
445     scp -i $KEYS_PATH/$KEY.pem $FILES_PATH/run-$benchmark.sh
... root@${VM_PUB_DNS[$COUNT]}:/root/>/dev/null 2>&1

```

```

446     ssh -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]} "chmod 755 /root/run-$benchmark.sh"
... >/dev/null 2>&1
447     echo
... #####
... ##### >>$TMP_PATH/launch_logs/run_error_`printf %03d
... $COUNT`.log
448     echo "`date` ${VM_IDS[$COUNT]}:${I_TYPES[$COUNT]}:$benchmark"
... >>$TMP_PATH/launch_logs/run_error_`printf %03d $COUNT`.log
449     ssh -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]} "/root/run-$benchmark.sh $TIME
... $COUNT $PARAM">/dev/null 2>>$TMP_PATH/launch_logs/run_error_`printf %03d $COUNT`.log 2>&1 &
450 done
451
452 echo "I'm done. Wait for the benchmark to be over and than launch the crop_results.sh script."
453
454 exit
455
456
457
458 -----
459 -----
460 -----
461
462
463 File: predictiondecisionengine/trunk/bin/aws_login.sh
464 #!/bin/bash
465 . /home/crivella/bin/functions.sh
466
467 while getopts hz: opt; do
468     case $opt in
469         z) if [ [ `grep "profile $OPTARG" $HOME/.aws/config` != "" ] ]; then
470             ZONE="--profile $OPTARG"
471         else
472             echo "You don't have a profile to work with this zone. Exiting..."
473             exit
474         fi
475     ;;
476     h | \?) cat<<E0U
477

```

```

478 Usage: $0 [z zone]
479 -z zone          Region to launch the instances in
480 EOU
481     exit
482     ;;
483 esac
484 done
485
486 FILES_PATH=/home/crivella/bin/Fermilab
487 KEYS_PATH=/home/crivella/Fermilab
488 USER=
489
490 #Check for registered user
491 USER_DATA=`aws iam get-user`
492 USER_ARN=`echo "$USER_DATA" | grep Arn | cut -d "\"" -f 4`
493 if [[ -z "$USER_ARN" ]]; then
494     echo "You don't appear to be registered to aws services. Exiting"
495 else
496     USER=`echo "$USER_DATA" | grep UserName | cut -d "\"" -f 4`
497     USER_ID=`echo "$USER_ARN" | cut -d ":" -f 5`
498     #echo $USER:$USER_ID
499 fi
500
501 VMS_DATA=`aws ec2 describe-instances $ZONE --filters Name=tag-value,Values=$USER
... Name=instance-state-name,Values=running`
502 if [[ `echo "$VMS_DATA" | grep $USER` = "" ]]; then
503     echo "No running instances found for user $USER. Quitting script..."
504     exit
505 fi
506
507 VMS=`echo "$VMS_DATA" | grep -F "$USER-" | cut -d "\"" -f 4`
508 A_VMS=$(list_to_array "$VMS")
509 unset VMS
510 VMS_KEYS=`echo "$VMS_DATA" | grep -F "$USER" | cut -d "\"" -f 4`
511 A_VMS_KEYS=$(list_to_array "$VMS_KEYS")
512 unset VMS_KEYS
513 VMS_PUB_DNS=`echo "$VMS_DATA" | grep -F "PublicDnsName" | cut -d "\"" -f 4 | uniq`
514 A_VMS_PUB_DNS=$(list_to_array "$VMS_PUB_DNS")

```

```

515 unset VMS_PUB_DNS
516
517 #Sort the Vms by name while also doing the same operation on the VMS_KEYS and VMS_PUB_DNS
... arrays
518 for ((i=0;i<${#A_VMS[@]}-1;i++)); do
519     for ((j=i+1;j<${#A_VMS[@]};j++)); do
520         if [[ "${A_VMS[$i]}" > "${A_VMS[$j]}" ]]; then
521             APP=${A_VMS[$i]}
522             A_VMS[$i]=${A_VMS[$j]}
523             A_VMS[$j]=$APP
524             APP=${A_VMS_KEYS[$i]}
525             A_VMS_KEYS[$i]=${A_VMS_KEYS[$j]}
526             A_VMS_KEYS[$j]=$APP
527             APP=${A_VMS_PUB_DNS[$i]}
528             A_VMS_PUB_DNS[$i]=${A_VMS_PUB_DNS[$j]}
529             A_VMS_PUB_DNS[$j]=$APP
530         fi
531     done
532 done
533
534 make_list "${A_VMS[@]}"
535 read_choice 5 `seq $LENGHT`
536 let CHOICE--
537
538 VM=${A_VMS[$CHOICE]}
539 VM_DNS=${A_VMS_PUB_DNS[$CHOICE]}
540 VM_KEY=${A_VMS_KEYS[$CHOICE]}
541
542 KEY=`find $KEYS_PATH -name "$VM_KEY.pem"`
543 if [ "$KEY" = "" ]; then
544     echo "You do not have the necessary key. Terminating..."
545     exit
546 fi
547
548 :
549 echo "Do you want to copy the files?[y/n]: "
550 read ANSWER
551 if [[ $ANSWER = "y" ]]; then

```

```

552 | scp -i $KEY $FILES_PATH/runttbar.sh root@$VM_DNS:/root/
553 | scp -i $KEY $FILES_PATH/res_show.sh root@$VM_DNS:/root/
554 | scp -i $KEY $FILES_PATH/runhepspec.sh root@$VM_DNS:/root/
555 | #scp -i $KEY $FILES_PATH/runparallel.sh root@$VM_DNS:/root/
556 | fi'
557 |
558 | ssh -i $KEY root@$VM_DNS
559 |
560 | -----
561 | -----
562 | -----
563 |
564 |
565 | File: predictiondecisionengine/trunk/bin/crop_results.sh
566 | #!/bin/bash
567 | . /home/crivella/bin/functions.sh
568 |
569 |
570 | #Check if needed packages are installed
571 | if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
572 |     echo "This script require aws-cli to run."
573 |     echo "Follow the instructions on https://aws.amazon.com/cli/ on how to get it"
574 |     exit
575 |
576 | fi
577 | if [[ `which nmap` = "" || `which nmap | grep -F "no nmap" != "" ]]; then
578 |     echo "This script require nmap to work. Install it by writing sudo apt-get install nmap"
579 |     echo "...Or the equivlaent command for your distribution"
580 |     exit
581 | fi
582 |
583 | #Edit this part with your data
584 | #Path where you wish to store the results of the benchmarks
585 | B_PATH=/home/crivella/Fermilab/Benchmarks
586 | #Your username
587 | USER=grassano
588 | #Where you have the .sh files
589 | FILES_PATH=/home/crivella/bin/Fermilab

```

```

590 | #Where you stored your .pem keys
591 | KEYS_PATH=/home/crivella/Fermilab/Keys
592 | #They name of the key you used without the .pem
593 | KEY=usw_oregon_grassano
594 | #where to keep the record of the machines you launched
595 | TMP_PATH=/home/crivella/Fermilab/tmp
596 |
597 | while getopts hk:z: opt; do
598 |     case $opt in
599 |         z) if [[ `grep "profile $OPTARG" $HOME/.aws/config` != "" ]]; then
600 |             ZONE="--profile $OPTARG"
601 |         else
602 |             echo "You don't have a profile to work with this zone. Exiting..."
603 |             exit
604 |         fi
605 |         ;;
606 |         k) KEY=$OPTARG
607 |         ;;
608 |         h | \?) cat<<EQU
609 |
610 | Usage: $0 [-k KEY_NAME] [z zone]
611 |    -k KEY_NAME      Specify the name of the key to use without .pem
612 |    -z zone          Region to launch the instances in
613 | EQU
614 |         exit
615 |         ;;
616 |     esac
617 | done
618 |
619 | declare -a benchmarks=(tt_bar_gensim tt_bar_reco hepspec06 s3_stresstest_d s3_stresstest_u
...
620 | test_fermigrid test_fermigrid2)
621 |
622 | echo "Chose the kind of benchmarks to crop the results for:"
623 |
624 | make_list "${benchmarks[@]}"
625 | read_choice 5 `seq $LENGHT`
626 | let CHOICE--

```

```

627 benchmark=${benchmarks[$CHOICE]}
628 mkdir -p $B_PATH/$benchmark
629
630 if [[ -d $TMP_PATH/$benchmark ]]; then
631     echo "I've found a tmp folder for the benchmark. Do you wish to use VM data from here?(y)"
632     echo "... Or do yo wish to query amazon for active VMS? (n)"
633     read_choice 5 "y" "n" "yes" "no"
634     FLAG=1
635 fi
636
637 if [[ "$CHOICE" != "n" && FLAG -eq 1 ]]; then
638     while read -r line; do
639         POS=`echo "$line" | cut -d ":" -f 1`
640         VMS[$POS]="`echo "$line" | cut -d ":" -f 2`"
641         VM_IDS[$POS]="`echo "$line" | cut -d ":" -f 3`"
642         VM_PUB_DNS[$POS]="`echo "$line" | cut -d ":" -f 4`"
643         VM_KEYS[$POS]="`echo "$line" | cut -d ":" -f 5`"
644     done < $TMP_PATH/$benchmark/VMS
645 else
646     #Adjust this section with your own format used to name instances and keys#####
647     VMS_DATA=`aws ec2 describe-instances $ZONE --filters Name=tag-value,Values=$USER`
648     VMS_NAME=`echo "$VMS_DATA" | grep -F "$USER-" | cut -d "\"" -f 4`
649     PUB_DNS=`echo "$VMS_DATA" | grep -F "PublicDnsName" | cut -d "\"" -f 4 | uniq`
650     VMS_KEYS=`echo "$VMS_DATA" | grep -F "_$USER" | cut -d "\"" -f 4`
651     #####
652
653     VM_PUB_DNS=$(list_to_array "$PUB_DNS")
654     VMS=$(list_to_array "$VMS_NAME")
655     VM_KEYS=$(list_to_array "$VMS_KEYS")
656     unset VMS_DATA
657     unset PUB_DNS
658     unset VMS_KEYS
659 fi
660
661 if [[ "${VM_PUB_DNS[0]}" = "" ]]; then
662     echo "No results found..."
663     exit
664 fi

```

```

665 : '
666 for ((COUNT=0;COUNT<${#VM_PUB_DNS[@]};COUNT++)); do
667     echo "$COUNT"
668     echo "${VMS[$COUNT]}"
669     echo "${VM_IDS[$COUNT]}"
670     echo "${VM_PUB_DNS[$COUNT]}"
671 done
672 exit'
673
674
675 if [[ "$benchmark" = "tt_bar_reco" || "$benchmark" = "tt_bar_gensim" ]]; then
676     cat <<CROP > crop.sh
677     #!/bin/bash
678
679     cd /tmp/cms
680     FOLDERS=`ls | grep results | grep -v .txt | grep -v .sh`
681
682     for CMD in \${FOLDERS}; do
683         if [[ -e check_results.sh ]]; then
684             sh check_results.sh > $CMD.txt
685         else
686             cd \${CMD}
687             sh ../check_results_gensim.sh > ../\${CMD}.txt
688             cd ..
689         fi
690     done
691     exit
692     CROP
693     fi
694
695     for ((COUNT=0;COUNT<${#VM_PUB_DNS[@]};COUNT++)); do
696         if [[ ((`echo "$benchmark" | grep "s3_stresstest" = "" && `echo "$benchmark" | grep
...
"test_fermigrid" = "")) || $COUNT -eq 0)) || ((`echo "$benchmark" | grep "s3_stresstest" != ""
...
|| `echo "$benchmark" | grep "test_fermigrid" != "")) && $COUNT -gt 0 && "${VMS[$COUNT]}" !=
...
"${VMS[(expr $COUNT - 1)]}" )); then
697             F_PATH="$B_PATH/$benchmark/${VMS[$COUNT]}"
698             A_PATH=$F_PATH
699             C_COUNT=0

```

```

700     while (! mkdir $A_PATH 2>/dev/null); do
701         let C_COUNT++
702         A_PATH=$F_PATH.run`printf %03d $C_COUNT`
703     done
704     F_PATH=$A_PATH
705 fi
706 #echo $F_PATH
707 if [[ ("benchmark" = "tt_bar_reco" || "benchmark" = "tt_bar_gensim") && $CHECK -eq 0 ]];
... then
708     scp -i $KEYS_PATH/${VM_KEYS[$COUNT]}.pem crop.sh
root@${VM_PUB_DNS[$COUNT]}:/root/>/dev/null
709     ssh -i $KEYS_PATH/${VM_KEYS[$COUNT]}.pem root@${VM_PUB_DNS[$COUNT]} "sh
... /root/crop.sh">/dev/null
710     #ssh -i $KEYS_PATH/${VM_KEYS[$COUNT]}.pem root@${VM_PUB_DNS[$COUNT]} "rm
... /root/crop.sh">/dev/null
711     scp -i $KEYS_PATH/${VM_KEYS[$COUNT]}.pem
root@${VM_PUB_DNS[$COUNT]}:/tmp/cms/results*txt $F_PATH>/dev/null
712     elif [ "benchmark" = "hepspec06" ]; then
713         #rm -f crop.sh
714         scp -i $KEYS_PATH/${VM_KEYS[$COUNT]}.pem -r
root@${VM_PUB_DNS[$COUNT]}:/scratch/install/results/* $F_PATH
715         #scp -r root@${VM_PUB_DNS[$COUNT]}:/scratch/new_hepspec2006/install/results/* $F_PATH
716         #exit
717         RESULTS=`find $F_PATH -name "lock.CPU2006" | sort`
718         if [ "$RESULTS" = "" ]; then
719             echo "The results folder is empty..."
720             exit
721         fi
722         while read -r CMD; do
723             RES_PATH=`dirname $CMD`
724             N_CORE=`find $RES_PATH -name "CPU2006*.log" | wc -l`
725             for COUNT in `seq $N_CORE`; do
726                 if [ $COUNT -eq 1 ]; then
727                     echo "CORE$COUNT" > $RES_PATH/crop_core.txt
728                 else
729                     echo -e "\nCORE$COUNT" >> $RES_PATH/crop_core.txt
730                 fi
731                 cat $RES_PATH/CFP2006.`printf %03d $COUNT`.ref.txt | grep -F " *" | sort |

```

```

731... uniq >> $RES_PATH/crop_core.txt
732         cat $RES_PATH/CINT2006.`printf %03d $COUNT`.ref.txt | grep -F " *" | sort |
...
733     done
734     done <<< "$RESULTS"
735     elif [ `echo "benchmark" | grep "s3_stresstest" != "" || `echo "benchmark" | grep
... "test_fermigrid" != "" ]; then
736         rm -f crop.sh
737         if [ [ "${VMS[$COUNT]}" = "${VMS[(expr $COUNT - 1)]}" && $COUNT -gt 0 ]; then
738             let I_COUNT++
739         else
740             I_COUNT=1
741         fi
742         if [ `echo "benchmark" | grep "test_fermigrid" != "" ]; then
743             scp -i $KEYS_PATH/${VM_KEYS[$COUNT]}.pem
... root@${VM_PUB_DNS[$COUNT]}:/scratch/fermigrid_upload.log $F_PATH/${VMS[$COUNT]}-$I_COUNT.log
744             scp -i $KEYS_PATH/${VM_KEYS[$COUNT]}.pem
... root@${VM_PUB_DNS[$COUNT]}:/scratch/command.log $F_PATH/command-$I_COUNT.log
745         else
746             scp -i $KEYS_PATH/${VM_KEYS[$COUNT]}.pem
root@${VM_PUB_DNS[$COUNT]}:/scratch/s3_stress.log $F_PATH/${VMS[$COUNT]}-$I_COUNT.log
747         fi
748         RESULTS="$F_PATH/${VMS[$COUNT]}-$I_COUNT.log"
749         while read -r CMD; do
750             RES_PATH=`dirname $CMD`
751             string=`cat $CMD | grep -F ":"`
752             #echo "$string"
753             C_COUNT=0
754             string2=`cat $CMD | grep -F "Simultaneous" | cut -d " " -f 4`
755             #echo "$string2"
756             N_COUNT=0
757             while read -r APP; do
758                 NUM[$N_COUNT]=$APP
759                 let N_COUNT++
760             done <<< "$string2"
761             string2=`cat $CMD | grep -F "out of" | cut -d " " -f 1`
762             #echo "$string2"
763             N_COUNT=0

```



```

838 |-----
839 |-----
840
841
842 |File: predictiondecisionengine/trunk/bin/Fermilab/From_EBS/aws_launch_benchmark.sh
843 |#!/bin/bash
844
845 |#Description: Read the choice to a query and confront it with a list of correct answer
... |limiting the numbers of checks
846 |#Parameters: 1st parameter: number of attempts possible other parameter:List of correct
... |answers
847 |function read_choice()
848 |{
849 |    local NUM=$1
850 |    shift
851 |    CHOICES={"${@}" }
852 |    while "true"; do
853 |        read CHOICE
854 |        CHOICE=`echo $CHOICE | tr YESNO yesno`
855 |        for CMD in ${CHOICES[@]}; do
856 |            if [[ "$CHOICE" = "$CMD" ]]; then
857 |                return 0
858 |            fi
859 |        done
860 |        echo "Choice is out of range. Enter again : "
861 |        let C_COUNT++
862 |        if [[ $C_COUNT -ge $NUM ]]; then
863 |            echo "$NUM invalid choices. Exiting..."
864 |            exit
865 |        fi
866 |    done
867 |}
868
869 |#Description: Make a numbered list out of an array or list of elements and return the number
... |of elements in LENGHT
870 |#Parameters: array or list of elements
871 |function make_list()
872 |{

```

```

873 |    local C_COUNT=0
874 |    if [[ `echo "$1" | wc -l` -gt 1 ]]; then
875 |        echo Lista
876 |        while read -r line; do
877 |            let C_COUNT++
878 |            echo -e "$C_COUNT-\t$line"
879 |        done <<< "$1"
880 |    else
881 |        local array={"${@}" }
882 |        for CMD in ${array[@]}; do
883 |            let C_COUNT++
884 |            echo -e "$C_COUNT-\t$CMD"
885 |        done
886 |    fi
887 |    LENGHT=$C_COUNT
888 |}
889
890 |#Check if needed packages are installed
891 |if [[ `which aws` = "" || `which aws | grep -F "no aws"` != "" ]]; then
892 |    echo "This script require aws-cli to run."
893 |    echo "Follow the instructions on https://aws.amazon.com/cli/ on how to get it"
894 |    exit
895 |fi
896 |
897 |if [[ `which nmap` = "" || `which nmap | grep -F "no nmap"` != "" ]]; then
898 |    echo "This script require nmap to work. Install it by writing sudo apt-get install nmap"
899 |    echo "...Or the equivlaent command for your distribution"
900 |    exit
901 |fi
902
903 |re='^[0-9]+$'
904
905 |#Edit this part with your data
906 |#Where you have the run-*.sh files
907 |FILES_PATH=$HOME/bin/Fermilab
908 |#Where you stored your .pem keys
909 |KEYS_PATH=$HOME/Fermilab/Keys
910 |#where to keep the record of the machines you launched

```

```

911 TMP_PATH=$HOME/Fermilab/tmp
912
913 #Your username
914 USER=grassano
915 #ID of the AMI you will be using
916 AMI_ID=ami-03dec833
917 #Just the name with no .pem, or modify rest of the script
918 KEY=usw_oregon_grassano
919 #ID of the security group
920 SG_ID=sg-91d414f5
921 #Not the same for every instance
922 SAME_INSTANCE_LIMIT=25
923
924 #Benchmark that the script can run
925 declare -a benchmarks=(tt_bar_gensim tt_bar_reco hepspec06 s3_stresstest_d s3_stresstest_u
... test_fermigrid)
926 #Volume to add with required size for the benchmark
927 declare -a req_vol_size=(80 80 50 120 3 3)
928 #Role related profile
929 declare -a profiles=(""" "" " AllowS3_Download AllowS3_Upload "AllowS3_Download")
930 declare -a parameters=(""" "" "" "1 10 100" "1 10 100" "1 5 10 20")
931
932 #Check for registered user
933 USER_DATA=`aws iam get-user`
934 USER_ARN=`echo "$USER_DATA" | grep Arn | cut -d "\"" -f 4`
935 if [[ -z "$USER_ARN" ]]; then
936     echo "You don't appear to be registered to aws services. Exiting"
937 else
938     USER=`echo "$USER_DATA" | grep UserName | cut -d "\"" -f 4`
939     USER_ID=`echo "$USER_ARN" | cut -d ":" -f 5`
940     #echo $USER:$USER_ID
941 fi
942
943 declare -a instances=(1:r3.xlarge 1:r3.2xlarge 1:r3.4xlarge)
944 RESUME_STATE="false"
945 while getopts hra:s:k:i:p: opt; do
946     case $opt in
947         a) AMI_ID=$OPTARG

```

```

948         ;;
949         s) SG_ID=$OPTARG
950         ;;
951         k) KEY=$OPTARG
952         ;;
953         i) unset instances
954             C_COUNT=0
955             for CMD in $OPTARG; do
956                 if [[ ! `echo $CMD | cut -d ":" -f 1` =~ $re || `echo $CMD | grep ":" -o | wc -l` -gt 1
... ]]; then
957                     echo "Invalid parameter for -i. Use the -h formore info."
958                     exit
959                 fi
960                 instances[$C_COUNT]=$CMD
961                 let C_COUNT++
962             done
963
964         ;;
965         p) PARAM=$OPTARG
966         ;;
967         r) RESUME_STATE="true"
968         if [[ "$OPTARG" != "" ]]; then
969             echo "The -r does not take additional parameter. Use the -h for additional info"
970             exit
971         fi
972         ;;
973         h | \?) cat<<EQU
974
975 Usage: $0 [-a AMI_ID] [-s Security_group_ID] [-k KEY_NAME] [-i instances] [-r] [-p parameters]
976 -a AMI_ID ID of the AMI to use
977 -s Security_group_ID ID of the security group to use
978 -k KEY_NAME Specify the name of the key to use without .pem
979 -i instances List of instances to start es: "3:m3.medium 2:c4.2xlarge"
980 -p parameters Parameters to pass to the run-*.sh script as "... .. ."
981 -r Enable resume mode
982 EQU
983     exit
984     ;;

```

```

985     esac
986 done
987
988 #Check if the selected AMI is available, if not give option to chose from list of owned
... available AMIs
989 while [[ `aws ec2 describe-images --image-id $AMI_ID 2>&1 | grep "\"Name\":" = "" ]]; do
990     echo "The specified ami does not exist. Do you wish to choose one from the ones you
... own?(y/n)"
991     read_choice 5 "y" "n" "yes" "no"
992     if [[ `echo $CHOICE | grep y` = "" ]]; then
993         exit
994     else
995         IMAGES=`aws ec2 describe-images --filters Name=state,Values=available --owners
... $USER_ID`
996         IMAGES_NAME=`echo "$IMAGES" | grep "\"Name\":" | cut -d "\"" -f 4`
997         IMAGES_ID=`echo "$IMAGES" | grep "\"ImageId\":" | cut -d "\"" -f 4`
998         make_list "$IMAGES_NAME"
999         read_choice 5 `seq $LENGHT`
1000         AMI_ID=`echo "$IMAGES_ID" | head -n $CHOICE | tail -n 1`
1001         #echo $AMI_ID
1002     fi
1003     unset IMAGES
1004     unset IMAGES_NAME
1005     unset IMAGES_ID
1006 done
1007
1008 #Check if the chosen key is available and recognized by aws
1009 while [[ `find $KEYS_PATH -name "$KEY.pem" = "" || `aws ec2 describe-key-pairs | grep $KEY` =
... "" ]]; do
1010     echo "You dont own the specified key or it is not recognized by amazon."
1011     echo "Do you wish to choose it from the ones listed in aws that you own as well?(y/n)"
1012     read_choice 5 "y" "n" "yes" "no"
1013     if [[ `echo $CHOICE | grep y` = "" ]]; then
1014         exit
1015     else
1016         KEYS=`find $KEYS_PATH -name "*.pem"`
1017         KEYS_DATA=`aws ec2 describe-key-pairs`
1018         C_COUNT=0

```

```

1019     while read -r line; do
1020         line=${line::-4}
1021         FIELD=`echo $line | grep "/" -o | wc -l`
1022         let FIELD++
1023         KEY=`echo $line | cut -d "/" -f $FIELD`
1024         if [[ `echo "$KEYS_DATA" | grep "$KEY" != "" ]]; then
1025             KEY_LIST[$C_COUNT]=$KEY
1026             let C_COUNT++
1027         fi
1028     done <<< "$KEYS"
1029     if [[ $C_COUNT -lt 1 ]]; then
1030         echo "No matching key found. Exiting..."
1031         exit
1032     fi
1033     make_list "${KEY_LIST[@]}"
1034     read_choice 5 `seq $LENGHT`
1035     let CHOICE--
1036     KEY="${KEY_LIST[$CHOICE]}"
1037     #echo $KEY
1038     fi
1039     unset KEY_LIST
1040     unset KEYS
1041     unset KEYS_DATA
1042 done
1043
1044 while [[ `aws ec2 describe-security-groups --group-ids $SG_ID 2>&1 | grep "\"GroupId\":" = ""
... ]]; do
1045     echo "The specified security group does not exist.Do you wish to chose one from those
... available on aws?(y/n)"
1046     read_choice 5 "y" "n" "yes" "no"
1047     if [[ `echo $CHOICE | grep y` = "" ]]; then
1048         exit
1049     else
1050         SGS=`aws ec2 describe-security-groups`
1051         SGS_NAME=`echo "$SGS" | grep "\"GroupName\":" | cut -d "\"" -f 4`
1052         echo "$SGS_NAME"
1053         SGS_ID=`echo "$SGS" | grep "\"GroupId\":" | cut -d "\"" -f 4`
1054         make_list "$SGS_NAME"

```

```

1055         read_choice 5 `seq $LENGHT`
1056         SG_ID=`echo "$SGS_ID" | head -n $CHOICE | tail -n 1`
1057         echo $SG_ID
1058     fi
1059     unset SGS
1060     unset SGS_NAME
1061     unset SGS_ID
1062 done
1063
1064 mkdir -p error_logs
1065
1066 echo "Chose the kind of benchmarks to execute:"
1067
1068 make_list "${benchmarks[@]}"
1069 read_choice 5 `seq $LENGHT`
1070 let CHOICE--
1071
1072 benchmark=${benchmarks[$CHOICE]}
1073 vol_size=${req_vol_size[$CHOICE]}
1074 if [[ -z "$PARAM" ]]; then
1075     PARAM=${parameters[$CHOICE]}
1076 fi
1077
1078 #If a profile is associated with the benchmark, enable it for the run-instances command
1079 if [[ "${profiles[$CHOICE]}" != "" ]]; then
1080     profile="--iam-instance-profile Name=\"${profiles[$CHOICE]}\"\"
1081 else
1082     profile=""
1083 fi
1084
1085
1086 #echo "$benchmark:$vol_size:$profile"
1087 #echo "${instances[@]}"
1088
1089 #####
1090 #Find an ebs volume with the same name of the benchmark containing the files. If not terminate
1091 #script

```

```

1091 #This script wont work if all the require files aren't in an ebs volume with the Name tag set
1092 #as the benchmark name
1093 #Exception for s3_stresstest_d/u that go with the same volume s3_stresstest
1094
1095 if [[ `echo $benchmark | grep s3_stresstest` != "" || "$benchmark" = "test_fermigrid" ]]; then
1096     a_benchmark="s3_stresstest"
1097 else
1098     a_benchmark=$benchmark
1099 fi
1100
1101 BM_VOLUME_DATA=`aws ec2 describe-volumes --filters Name=tag-value,Values=$a_benchmark`
1102 BM_VOLUME_ID=`echo "$BM_VOLUME_DATA" | grep -F "VolumeId" | cut -d "\"" -f 4`
1103 AVAL_ZONE=`echo "$BM_VOLUME_DATA" | grep -F "AvailabilityZone" | cut -d "\"" -f 4`
1104
1105 if [ "$BM_VOLUME_ID" = "" ]; then
1106     echo "Can't find the right volume. Quitting..."
1107     exit
1108 fi
1109
1110 #echo -e "$instances\n$PARAM\n$benchmark:$vol_size"
1111 #exit
1112 #####
1113 #Create folder to store the VMS data + Resume work option
1114 COUNT2=0
1115 RESUME=0
1116 mkdir -p $TMP_PATH/$benchmark
1117 if $RESUME_STATE; then
1118     if [[ -f $TMP_PATH/$benchmark/VMS ]]; then
1119         echo "I've found previously loaded tmp data. Do you wish to resume work?(y/n)"
1120         echo "(Works for scripts that got stopped before the volume creation section or
1121         requires manual bypass of script parts)"
1122         read_choice 5 "y" "n" "yes" "no"
1123     fi
1124     if [[ "$CHOICE" != "n" ]]; then
1125         while read -r line; do
1126             if [[ "$line" != "" ]]; then

```

```

1126 |         I_TYPES[`echo "$line" | cut -d ":" -f 1`]=`echo "$line" | cut -d ":" -f 2`
1127 |         VM_IDS[$COUNT2]=`echo "$line" | cut -d ":" -f 3`
1128 |         VM_PUB_DNS[$COUNT2]=`echo "$line" | cut -d ":" -f 4`
1129 |         #echo "$COUNT2 ${I_TYPES[$COUNT2]} ${CM_IDS[$COUNT2]}
... | ${VM_PUB_DNS[$COUNT2]}"
1130 |         let COUNT2++
1131 |         let RESUME++
1132 |     fi
1133 | done < $TMP_PATH/$benchmark/VMS
1134 | else
1135 |     echo "" > $TMP_PATH/$benchmark/VMS
1136 | fi
1137 | fi
1138 |
1139 | #####
... | #####
1140 | #Starting the required VM
1141 |
1142 | for ((COUNT=0;COUNT<${#instances[@]};COUNT++)); do
1143 |     I_NUM=`echo "${instances[$COUNT]}" | cut -d ":" -f 1`
1144 |     I_TYPE=`echo "${instances[$COUNT]}" | cut -d ":" -f 2`
1145 |
1146 |     if [[ $I_NUM -gt $SAME_INSTANCE_LIMIT ]]; then
1147 |         echo "The number ($I_NUM) specified for instance $I_TYPE exceeds the limit of
... | $SAME_INSTANCE_LIMIT."
1148 |         echo "Bringing it down to the maximun allowed..."
1149 |         I_NUM=$SAME_INSTANCE_LIMIT
1150 |     fi
1151 |     echo "Launching $I_NUM $I_TYPE instances"
1152 |
1153 |     INSTANCE_DATA=`aws ec2 run-instances --count $I_NUM --image-id $AMI_ID --instance-type
... | $I_TYPE --security-group-ids $SG_ID --placement
... | AvailabilityZone=$AVAL_ZONE,GroupName="",Tenancy=default --key-name $KEY $profile`
1154 |
1155 |     if [[ $RESUME -gt 0 && "${I_TYPES[${expr $RESUME - 1}]}" = "$I_TYPE" ]]; then
1156 |         I_COUNT=$RESUME
1157 |     else
1158 |         I_COUNT=1

```

```

1159 |     fi
1160 |     L_COUNT=1
1161 |     while [[ $I_COUNT -le $I_NUM ]]; do
1162 |         I_TYPES[$COUNT2]=$I_TYPE
1163 |         INSTANCE_ID=`echo "$INSTANCE_DATA" | grep -F "InstanceId" | head -n $L_COUNT | tail -n
... | 1 | cut -d "\"" -f 4`
1164 |         VM_IDS[$COUNT2]=$INSTANCE_ID
1165 |         aws ec2 create-tags --resources $INSTANCE_ID --tags
... | Key=Name,Value=$USER-$I_TYPE-$benchmark-$I_COUNT Key=User,Value=$USER>/dev/null
1166 |         let I_COUNT++
1167 |         let L_COUNT++
1168 |         let COUNT2++
1169 |     done
1170 | done
1171 | LAST_INSTANCE=$(expr $COUNT2 - 1)
1172 |
1173 | #####
... | #####
1174 | #Get the public DNS after they are available
1175 | echo "Waiting for the DNS..."
1176 | while [[ `aws ec2 describe-instances --instance-ids ${VM_IDS[$LAST_INSTANCE]} | grep -F
... | "PublicDnsName" | cut -d "\"" -f 4 | uniq` = "" ]]; do
1177 |     sleep 10
1178 | done
1179 | sleep 5
1180 |
1181 | for ((COUNT=$RESUME;COUNT<${#VM_IDS[@]};COUNT++)); do
1182 |     INSTANCE_DATA=`aws ec2 describe-instances --instance-ids ${VM_IDS[$COUNT]}`
1183 |     PUB_DNS=`echo "$INSTANCE_DATA" | grep -F "PublicDnsName" | cut -d "\"" -f 4 | uniq`
1184 |     #Print VMS data in tmp file
1185 |     echo "$COUNT:${I_TYPES[$COUNT]}:${VM_IDS[$COUNT]}:PUB_DNS" >> $TMP_PATH/$benchmark/VMS
1186 |     VM_PUB_DNS[$COUNT]=$PUB_DNS
1187 | done
1188 |
1189 | #####
... | #####
1190 | #Create extra volumes
1191 | for ((COUNT=0;COUNT<${#VM_IDS[@]};COUNT++)); do

```

```

1192     if [[ "${I_TYPES[$COUNT]}" = "${I_TYPES[(expr $COUNT - 1)]}" && $COUNT -gt 0 ]]; then
1193         let I_COUNT++
1194     else
1195         I_COUNT=1
1196     fi
1197     echo "Creating extradisk for instance ${I_TYPES[$COUNT]}-$I_COUNT"
1198     VOLUME_DATA=`aws ec2 create-volume --size $vol_size --availability-zone us-west-2a
... --volume-type gp2`
1199     VOLUME_ID[$COUNT]=`echo "$VOLUME_DATA" | grep -F "VolumeId" | cut -d "\"" -f 4`
1200 done
1201
1202 #####
... #####
1203 #Wait until all VM are done initialazing
1204 echo "Waiting for the instances to be initialized..."
1205 for ((COUNT=0;COUNT<${#VM_IDS[@]};COUNT++)); do
1206     while [[ `nmap -p 22 ${VM_PUB_DNS[$COUNT]} | grep open` = "" ]]; do
1207         sleep 15
1208     done
1209 done
1210
1211
1212 #####
... #####
1213 #Disable the self-stop service and register the connection to the DNS (won't ask yes/no
... afterwards)
1214 for ((COUNT=0;COUNT<${#VM_IDS[@]};COUNT++)); do
1215     ssh -o "StrictHostKeyChecking no" -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]}
... "service glideinwms-pilot stop"/dev/null
1216     #echo "Stopped service in instance ${instances[$COUNT]}."
1217 done
1218
1219
1220 #####
... #####
1221 #Waiting for volumes to create
1222 echo "Waiting for volumes to create..."
1223 for ((COUNT=0;COUNT<${#VOLUME_ID[@]};COUNT++)); do

```

```

1224     while [[ `aws ec2 describe-volumes --volume-ids ${VOLUME_ID[$COUNT]} | grep available` =
... "" ]]; do
1225         #echo "Waiting for disk to create..."
1226         sleep 5
1227     done
1228     if [[ "${I_TYPES[$COUNT]}" = "${I_TYPES[(expr $COUNT - 1)]}" && $COUNT -gt 0 ]]; then
1229         let I_COUNT++
1230     else
1231         I_COUNT=1
1232     fi
1233     aws ec2 create-tags --resources ${VOLUME_ID[$COUNT]} --tags
... Key=Name,Value=$USER-${I_TYPES[$COUNT]}-$benchmark-$I_COUNT Key=user,Value=$USER>/dev/null
1234 done
1235
1236 #####
... #####
1237 #Attaching Volumes
1238 echo "Attaching Volumes..."
1239 for ((COUNT=0;COUNT<${#VOLUME_ID[@]};COUNT++)); do
1240     aws ec2 attach-volume --volume-id ${VOLUME_ID[$COUNT]} --instance-id ${VM_IDS[$COUNT]}
... --device /dev/sdf>/dev/null
1241 done
1242
1243 #####
... #####
1244 #Waiting for volumes to attach
1245 echo "Waiting for volumes to attach and making filesystem..."
1246 for ((COUNT=0;COUNT<${#VOLUME_ID[@]};COUNT++)); do
1247     while [[ `aws ec2 describe-volumes --volume-ids ${VOLUME_ID[$COUNT]} | grep attaching` !=
... "" ]]; do
1248         sleep 3
1249     done
1250     if [[ "${I_TYPES[$COUNT]}" = "${I_TYPES[(expr $COUNT - 1)]}" && $COUNT -gt 0 ]]; then
1251         let I_COUNT++
1252     else
1253         I_COUNT=1
1254     fi
1255     echo "Making file system ext4 for ${I_TYPES[$COUNT]}-$I_COUNT..."

```

```

1256 ssh -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]} "mkfs.ext4 /dev/xvdf
... 2>/dev/null">/dev/null &
1257 done
1258 wait
1259 : '
1260 #Changed to create for all
1261 #####
... #####
1262 #If the instance doesn't have a storage volume create it and make the system file as ext4
1263 echo "Creating extrastorage if needed..."
1264 for ((COUNT=0;COUNT<${#VM_IDS[@]};COUNT++)); do
1265     #if [[ `echo "${instances[$COUNT]}" | grep 3` = "" ]]; then
1266         if [[ "${I_TYPES[$COUNT]}" = "${I_TYPES[(expr $COUNT - 1)]}" && $COUNT -gt 0 ]]; then
1267             let I_COUNT++
1268         else
1269             I_COUNT=1
1270         fi
1271         echo "Creating extradisk for instance ${I_TYPES[$COUNT]}-$I_COUNT"
1272         VOLUME_DATA=`aws ec2 create-volume --size $vol_size --availability-zone us-west-2a
... --volume-type gp2`
1273         VOLUME_ID=`echo "$VOLUME_DATA" | grep -F "VolumeId" | cut -d "\"" -f 4`
1274         aws ec2 create-tags --resources $VOLUME_ID --tags
... Key=Name,Value=$USER-${instances[$COUNT]}-$benchmark-$I_COUNT Key=user,Value=$USER>/dev/null
1275         while [[ `aws ec2 describe-volumes --volume-ids $VOLUME_ID | grep available` = "" ]]; do
1276             #echo "Waiting for disk to create..."
1277             sleep 5
1278         done
1279         aws ec2 attach-volume --volume-id $VOLUME_ID --instance-id ${VM_IDS[$COUNT]} --device
... /dev/sdf>/dev/null
1280         while [[ `aws ec2 describe-volumes --volume-ids $VOLUME_ID | grep attaching` != "" ]]; do
1281             sleep 3
1282         done
1283         ssh -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]} "mkfs.ext4 /dev/xvdf
... 2>/dev/null">/dev/null
1284         #echo "Volume attaccato a ${instances[$COUNT]}"
1285     #else
1286         #echo "No extradisk for instance ${instances[$COUNT]}."
1287     #fi

```

```

1288 done'
1289
1290 #####
... #####
1291 #Mount the volumes and transfer the benchmark via script
1292 echo "Mounting volumes and transferring benchmarks..."
1293 : 'cat <<TRANSFER > transfer.sh
1294 #!/bin/bash
1295
1296 mkdir -p /scratch
1297 mkdir -p /scratch02
1298
1299 if [[ `fdisk -l | grep xvdb` != "" ]]; then
1300     mount /dev/xvdb /scratch
1301     mount /dev/xvdf /scratch02
1302 else
1303     mount /dev/xvdf /scratch/
1304     mount /dev/xvdg /scratch02/
1305 fi
1306 cp /scratch02/* /scratch/
1307 if [[ `fdisk -l | grep xvdb` != "" ]]; then
1308     umount /dev/xvdf
1309 else
1310     umount /dev/xvdg
1311 fi
1312 rm -r -f /scratch02/
1313 exit
1314 TRANSFER'
1315
1316 cat <<TRANSFER > transfer.sh
1317 #!/bin/bash
1318
1319 mkdir -p /scratch
1320 mkdir -p /scratch02
1321
1322 mount /dev/xvdf /scratch
1323 mount /dev/xvdg /scratch02
1324

```

```

1325 cp /scratch02/* /scratch/
1326
1327 umount /dev/xvdg
1328
1329 rm -r -f /scratch02/
1330 exit
1331 TRANSFER
1332
1333 for ((COUNT=0;COUNT<${#VM_IDS[@]};COUNT++)); do
1334     scp -i $KEYS_PATH/$KEY.pem transfer.sh root@${VM_PUB_DNS[$COUNT]}:/root/>/dev/null 2>&1
1335     : 'if [[ `echo "${instances[$COUNT]}" | grep 3` = "" ]]; then
1336         aws ec2 attach-volume --volume-id $BM_VOLUME_ID --instance-id ${VM_IDS[$COUNT]}
1337     --device /dev/sdg>/dev/null
1338     else
1339         aws ec2 attach-volume --volume-id $BM_VOLUME_ID --instance-id ${VM_IDS[$COUNT]}
1340     --device /dev/sdf>/dev/null
1341     fi'
1342
1343     aws ec2 attach-volume --volume-id $BM_VOLUME_ID --instance-id ${VM_IDS[$COUNT]} --device
1344 /dev/sdg>/dev/null
1345 while [[ `aws ec2 describe-volumes --volume-ids $BM_VOLUME_ID | grep attaching` != "" ]];
1346 do
1347     sleep 3
1348 done
1349 ssh -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]} "sh /root/transfer.sh">/dev/null
1350 2>>error_logs/transfer_error-$COUNT.log
1351 ssh -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]} "rm /root/transfer.sh">/dev/null
1352 2>&1
1353 aws ec2 detach-volume --volume-id $BM_VOLUME_ID --instance-id ${VM_IDS[$COUNT]} --device
1354 /dev/sdg>/dev/null
1355 : 'if [[ `echo "${instances[$COUNT]}" | grep 3` = "" ]]; then
1356     aws ec2 detach-volume --volume-id $BM_VOLUME_ID --instance-id ${VM_IDS[$COUNT]}
1357 --device /dev/sdg>/dev/null
1358 else
1359     aws ec2 detach-volume --volume-id $BM_VOLUME_ID --instance-id ${VM_IDS[$COUNT]}
1360 --device /dev/sdf>/dev/null
1361 fi'

```

```

1354 while [[ `aws ec2 describe-volumes --volume-ids $BM_VOLUME_ID | grep detaching` != "" ]];
1355 do
1356     sleep 5
1357 done
1358 done
1359 rm -f transfer.sh
1360
1361 #####
1362 #####
1363 #Start the benchmark
1364 echo "Starting the benchmark on every instance..."
1365 for ((COUNT=0;COUNT<${#VM_IDS[@]};COUNT++)); do
1366     scp -i $KEYS_PATH/$KEY.pem $FILES_PATH/run-$benchmark.sh
1367     root@${VM_PUB_DNS[$COUNT]}:/root/>/dev/null 2>&1
1368     ssh -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]} "chmod 755 /root/run-$benchmark.sh"
1369     >/dev/null 2>&1
1370     ssh -i $KEYS_PATH/$KEY.pem root@${VM_PUB_DNS[$COUNT]} "/root/run-$benchmark.sh
1371 $PARAM">/dev/null 2>>error_logs/run_error_$COUNT.log &
1372 done
1373
1374 echo "I'm done. Wait for the benchmark to be over and than launch the crop_results.sh script."
1375
1376 exit
1377
1378 -----
1379 -----
1380
1381 File: predictiondecisionengine/trunk/bin/Fermilab/From_EBS/run-hepspec06.sh
1382 #!/bin/bash
1383
1384 if [[ `which rpm` = "" || `which rpm | grep -F "no aws"` != "" ]]; then
1385     yum install rpm

```

```

1387 fi
1388
1389 if [[ `rpm -qa gcc-c++` = "" ]]; then
1390     yum install gcc-c++
1391 fi
1392
1393 cd /scratch
1394 tar xvzf SPEC_CPU2006v1.1.tar.gz
1395
1396 cd SPEC_CPU2006v1.1
1397
1398 ln -s /usr/bin/gcc /usr/local/bin/gcc
1399 ./install.sh -d ../install/ -f
1400
1401 cd ..
1402 mkdir hepspec
1403 cd hepspec
1404 tar xvzf ../spec2k6-2.23.tar.gz
1405 cp linux64-gcc_cern.cfg ../install/config
1406
1407
1408 cd ../install
1409 . ./shrc
1410 #runspec --config=linux64-gcc_cern.cfg all_cpp
1411 mkdir -p ./results
1412 #mv ./result ./results/Single_core
1413
1414 COUNT=`grep -c "^processor" /proc/cpuinfo`;
1415 for i in `seq $COUNT`;
1416 do
1417     runspec --config=linux64-gcc_cern.cfg all_cpp &
1418 done
1419 wait
1420
1421 mv ./result ./results/All_cores
1422
1423
1424 -----

```

```

1425 -----
1426 -----
1427
1428
1429 File: predictiondecisionengine/trunk/bin/Fermilab/From_EBS/run-s3_stresstest_d.sh
1430 #!/bin/bash
1431
1432 TEST_BUCKET=grassano-test
1433 TEST_FILE=ACEE623F-B1A8-E411-8672-0025905938B4.root
1434 : 'service glideinwms-pilot stop
1435
1436 mkfs.ext4 /dev/xvdf 2>/dev/null
1437 mkdir -p /scratch
1438 mount /dev/xvdf /scratch'
1439
1440 cd /scratch
1441
1442 if [[ `which aws` = "" || `which aws | grep -F "no aws"` != "" ]]; then
1443     #wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
1444     unzip awscli-bundle.zip
1445     python awscli-bundle/install
1446     PATH=$PATH:/root/.local/lib/aws/bin
1447     export PATH
1448     #mkdir /root/.aws
1449     #mv /scratch/config /root/.aws/
1450     #chmod 400 /root/.aws/config
1451 fi
1452
1453
1454 echo "Download test" > s3_stress.log
1455 LIST="1 10 100"
1456 if [[ $# -gt 0 ]]; then
1457     LIST="$@"
1458 fi
1459 #NUM=1
1460 for NUM in $LIST; do
1461     mkdir -p $NUM
1462     echo "Simultaneous download of $NUM copies..." >>s3_stress.log

```

```

1463     date >>s3_stress.log
1464     for n in `seq $NUM`; do
1465         aws s3 cp s3://$TEST_BUCKET/$TEST_FILE $NUM/$n.root > $NUM/$n.log 2>&1 &
1466     done
1467     wait
1468     date >>s3_stress.log
1469     echo -e "`ls -l $NUM/ | grep 1073604531 | wc -l` out of $NUM successfully downloaded\n"
... >>s3_stress.log
1470     #let NUM*=10
1471     #echo $NUM
1472 done
1473
1474 exit
1475
1476 -----
1477 -----
1478 -----
1479
1480
1481 File: predictiondecisionengine/trunk/bin/Fermilab/From_EBS/run-s3_stresstest_u.sh
1482 #!/bin/bash
1483
1484 TEST_BUCKET=grassano-test
1485 TEST_FILE=ACEE623F-B1A8-E411-8672-0025905938B4.root
1486 : 'service glideinwms-pilot stop
1487
1488 mkfs.ext4 /dev/xvdf 2>/dev/null
1489 mkdir -p /scratch
1490 mount /dev/xvdf /scratch'
1491
1492 cd /scratch
1493
1494 if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
1495     #wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
1496     unzip awscli-bundle.zip
1497     python awscli-bundle/install
1498     PATH=$PATH:/root/.local/lib/aws/bin
1499     export PATH

```

```

1500     #mkdir /root/.aws
1501     #mv /scratch/config /root/.aws/
1502     #chmod 400 /root/.aws/config
1503 fi
1504
1505 ID=$RANDOM
1506 echo "Upload test ID=$ID" > s3_stress.log
1507 LIST="1 10 100"
1508 if [[ $# -gt 0 ]]; then
1509     LIST="$@"
1510 fi
1511 #NUM=1
1512 for NUM in $LIST; do
1513     mkdir -p $NUM
1514     echo "Simultaneous upload of $NUM copies..." >>s3_stress.log
1515     date >>s3_stress.log
1516     for n in `seq $NUM`; do
1517         aws s3 cp $TEST_FILE s3://$TEST_BUCKET/$NUM-$ID/$n.root > $NUM/$n.log 2>&1 &
1518     done
1519     wait
1520     date >>s3_stress.log
1521     echo -e "`aws s3api list-objects --bucket $TEST_BUCKET --prefix $NUM-$ID/ | grep
... 1073604531 | wc -l` out of $NUM successfully uploaded\n" >>s3_stress.log
1522     #let NUM*=10
1523     #echo $NUM
1524 done
1525
1526 exit
1527
1528 -----
1529 -----
1530 -----
1531
1532
1533 File: predictiondecisionengine/trunk/bin/Fermilab/From_EBS/run-test_fermigrid.sh
1534 #!/bin/bash
1535
1536 TEST_BUCKET=grassano-test

```

```

1537 TEST_FILE=ACEE623F-B1A8-E411-8672-0025905938B4.root
1538
1539 cd /scratch
1540 mv x509up_u50685 /tmp/x509up_u0
1541 voms-proxy-init -noregen -voms fermilab:/fermilab
1542 : '
1543 if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
1544     wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
1545     unzip awscli-bundle.zip
1546     python awscli-bundle/install
1547     PATH=$PATH:/root/.local/lib/aws/bin
1548     export PATH
1549     #mkdir /root/.aws
1550     #mv /scratch/config /root/.aws/
1551     #chmod 400 /root/.aws/config
1552 fi
1553 '
1554
1555 DEST=gsiftp://fndca1.fnal.gov:2811/fermigrid202/grassano
1556 ID=$RANDOM
1557 echo "Upload test ID=$ID" > fermigrid_upload.log
1558 echo "" > data.txt
1559 if [[ ! -f $TEST_FILE ]]; then
1560     aws s3 cp s3://$TEST_BUCKET/$TEST_FILE ./
1561 fi
1562 LIST="1 5 10 20"
1563 if [[ $# -gt 0 ]]; then
1564     LIST="$@"
1565 fi
1566 for NUM in $LIST; do
1567     #mkdir -p $NUM
1568     echo "Simultaneous upload of $NUM copies..." >>fermigrid_upload.log
1569     echo "" > data.txt
1570     for n in `seq $NUM`; do
1571         echo "file:///scratch/$TEST_FILE $DEST/$ID/$n.root" >> data.txt
1572     done
1573     date >>fermigrid_upload.log
1574     globus-url-copy -cd -fast -p 4 -cc $NUM -f data.txt >> globus.log 2>&1

```

```

1575     date >>fermigrid_upload.log
1576     echo -e "`edg-gridftp-ls -v $DEST/$ID/ | grep 1073604531 | wc -l` out of $NUM successfully
... uploaded\n" >>fermigrid_upload.log
1577     for n in `seq $NUM`; do
1578         edg-gridftp-rm $DEST/$ID/$n.root
1579     done
1580     edg-gridftp-rmdir $DEST/$ID/
1581     #let NUM*=10
1582     #echo $NUM
1583 done
1584
1585 exit
1586
1587 -----
1588 -----
1589 -----
1590
1591
1592 File: predictiondecisionengine/trunk/bin/Fermilab/From_EBS/run-tt_bar_gensim.sh
1593 #!/bin/bash
1594
1595 cd /scratch
1596
1597 tar -zxf benchmark-2015.tgz
1598
1599 groupadd cms_admin
1600 useradd -g cms_admin cms_admin
1601
1602 ln -s /scratch/cms/ /tmp/cms
1603 chown -R cms_admin.cms_admin /tmp/cms/
1604
1605 cd /tmp/cms/
1606 su cms_admin -c "sh benchmark-2015-gensim.sh 1">log
1607 mv results results.Single_core
1608
1609 su cms_admin -c "sh benchmark-2015-gensim.sh">log
1610 mv results results.All_cores
1611

```

```

1612 |-----
1613 |-----
1614 |-----
1615
1616
1617 File: predictiondecisionengine/trunk/bin/Fermilab/From_EBS/run-tt_bar_reco.sh
1618 #!/bin/bash
1619
1620 groupadd cms_admin
1621 useradd -g cms_admin cms_admin
1622
1623 cd /scratch
1624 tar -zxf /scratch/benchmark-2015.tgz
1625 ln -s /scratch/tmp/cms/ /tmp/cms
1626
1627 mv /scratch/ACEE623F-B1A8-E411-8672-0025905938B4.root /tmp/cms
1628 mv /scratch/ttbarGENSIM.root /tmp/cms
1629 mv /scratch/step2-50ns-4x-6400.root /tmp/cms
1630 mv -f /scratch/*.sh /tmp/cms
1631
1632 mv /scratch/frontier-cache.tgz /tmp/cms
1633 cd /tmp/cms/
1634 tar -zxf /tmp/cms/frontier-cache.tgz
1635
1636 chown -R cms_admin.cms_admin /tmp/cms/
1637
1638 su cms_admin -c "sh start_x_benchmarks_reco.sh">>log &
1639
1640 exit
1641
1642 |-----
1643 |-----
1644 |-----
1645
1646
1647 File: predictiondecisionengine/trunk/bin/Fermilab/run-hepspec06.sh
1648 #!/bin/bash
1649

```

```

1650 shift
1651 shift
1652
1653 TEST_BUCKET=grassano-test
1654
1655 mkdir -p /scratch/
1656 mount /dev/xvdf /scratch/
1657
1658 if [[ `which rpm` = "" || `which rpm | grep -F "no aws" != "" ]]; then
1659     yum install rpm
1660 fi
1661
1662 if [[ `rpm -qa gcc-c++` = "" ]]; then
1663     yum install gcc-c++
1664 fi
1665
1666 cd /scratch
1667
1668 if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
1669     if [[ -e /root/.local/lib/aws/bin ]]; then
1670         export PATH=$PATH:/root/.local/lib/aws/bin
1671     else
1672         wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
1673         unzip awscli-bundle.zip
1674         python awscli-bundle/install
1675         export PATH=$PATH:/root/.local/lib/aws/bin
1676         #mkdir /root/.aws
1677         #mv /scratch/config /root/.aws/
1678         #chmod 400 /root/.aws/config
1679     fi
1680 fi
1681
1682 aws s3 cp --recursive s3://$TEST_BUCKET/hepspec06/ .
1683
1684 tar xvzf SPEC_CPU2006v1.1.tar.gz
1685
1686 cd SPEC_CPU2006v1.1
1687

```

```

1688 ln -s /usr/bin/gcc /usr/local/bin/gcc
1689 ./install.sh -d ../install/ -f
1690
1691 cd ..
1692 mkdir hepspec
1693 cd hepspec
1694 tar xvzf ../spec2k6-2.23.tar.gz
1695 cp linux64-gcc_cern.cfg ../install/config
1696
1697
1698 cd ../install
1699 . ./shrc
1700 if [[ $# -eq 1 && $1 -eq 0 ]]; then
1701     exit
1702 fi
1703 runspec --config=linux64-gcc_cern.cfg all_cpp
1704 mkdir -p ./results
1705 mv ./result ./results/Single_core
1706
1707 COUNT=`grep -c "^processor" /proc/cpuinfo`;
1708 for i in `seq $COUNT`;
1709 do
1710     runspec --config=linux64-gcc_cern.cfg all_cpp &
1711 done
1712 wait
1713
1714 mv ./result ./results/All_cores
1715
1716
1717 -----
1718 -----
1719 -----
1720
1721
1722 File: predictiondecisionengine/trunk/bin/Fermilab/run-s3_stresstest_d.sh
1723 #!/bin/bash
1724
1725 shift

```

```

1726 shift
1727 TEST_BUCKET=grassano-test
1728 TEST_FILE=ACEE623F-B1A8-E411-8672-0025905938B4.root
1729
1730 mkdir -p /scratch/
1731 mount /dev/xvdf /scratch/
1732 cd /scratch
1733
1734 if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
1735     if [[ -e /root/.local/lib/aws/bin ]]; then
1736         export PATH=$PATH:/root/.local/lib/aws/bin
1737     else
1738         wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
1739         unzip awscli-bundle.zip
1740         python awscli-bundle/install
1741         export PATH=$PATH:/root/.local/lib/aws/bin
1742         #mkdir /root/.aws
1743         #mv /scratch/config /root/.aws/
1744         #chmod 400 /root/.aws/config
1745     fi
1746 fi
1747
1748 echo "Download test" > s3_stress.log
1749 LIST="1 10 100"
1750 if [[ $# -gt 0 ]]; then
1751     LIST="$@"
1752 fi
1753 if [[ $# -eq 1 && $1 -eq 0 ]]; then
1754     exit
1755 fi
1756 #NUM=1
1757 for NUM in $LIST; do
1758     mkdir -p $NUM
1759     echo "Simultaneous download of $NUM copies..." >>s3_stress.log
1760     date >>s3_stress.log
1761     for n in `seq $NUM`; do
1762         aws s3 cp s3://$TEST_BUCKET/s3_stresstest/$TEST_FILE $NUM/$n.root > $NUM/$n.log 2>&1 &
1763     done

```

```

1764     wait
1765     date >>s3_stress.log
1766     echo -e "`ls -l $NUM/ | grep 1073604531 | wc -l` out of $NUM successfully downloaded\n"
... >>s3_stress.log
1767     #let NUM*=10
1768     #echo $NUM
1769 done
1770
1771 exit
1772
1773 -----
1774 -----
1775 -----
1776
1777
1778 File: predictiondecisionengine/trunk/bin/Fermilab/run-s3_stresstest_u.sh
1779 #!/bin/bash
1780
1781 TIME=$1
1782 shift
1783 ID=$1
1784 shift
1785 sleep $((240 + `date -d "$TIME" +%s` - `date +%s`)) &
1786 WAIT_ID=$!
1787
1788 TEST_BUCKET=grassano-test
1789 TEST_FILE=ACEE623F-B1A8-E411-8672-0025905938B4.root
1790
1791 mkdir -p /scratch/
1792 mount /dev/xvdf /scratch/
1793 cd /scratch
1794
1795 if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
1796     if [[ -e /root/.local/lib/aws/bin ]]; then
1797         export PATH=$PATH:/root/.local/lib/aws/bin
1798     else
1799         wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
1800         unzip awscli-bundle.zip

```

```

1801     python awscli-bundle/install
1802     export PATH=$PATH:/root/.local/lib/aws/bin
1803     #mkdir /root/.aws
1804     #mv /scratch/config /root/.aws/
1805     #chmod 400 /root/.aws/config
1806 fi
1807 fi
1808
1809 aws s3 cp s3://$TEST_BUCKET/s3_stresstest/$TEST_FILE .
1810
1811 #ID=$RANDOM
1812 echo "Upload test ID=$ID" > s3_stress.log
1813 LIST="1 10 100"
1814 if [[ $# -gt 0 ]]; then
1815     LIST="$@"
1816 fi
1817 if [[ $# -eq 1 && $1 -eq 0 ]]; then
1818     exit
1819 fi
1820
1821 wait $WAIT_ID
1822 for NUM in $LIST; do
1823     mkdir -p $NUM
1824     echo "Simultaneous upload of $NUM copies..." >>s3_stress.log
1825     date >>s3_stress.log
1826     for n in `seq $NUM`; do
1827         aws s3 cp $TEST_FILE s3://$TEST_BUCKET/$NUM-$ID/$n.root > $NUM/$n.log 2>&1 &
1828     done
1829     wait
1830     date >>s3_stress.log
1831     echo -e "`aws s3api list-objects --bucket $TEST_BUCKET --prefix $NUM-$ID/ | grep
... 1073604531 | wc -l` out of $NUM successfully uploaded\n" >>s3_stress.log
1832     #let NUM*=10
1833     #echo $NUM
1834 done
1835
1836 exit
1837

```

```

1838 |-----
1839 |-----
1840 |-----
1841
1842
1843 File: predictiondecisionengine/trunk/bin/Fermilab/run-test_fermigrid.sh
1844 #!/bin/bash
1845
1846 TIME=$1
1847 shift
1848 ID=$1
1849 shift
1850 sleep $((240 + `date -d "$TIME" +%s` - `date +%s`)) &
1851 WAIT_ID=$!
1852
1853 TEST_BUCKET=grassano-test
1854 TEST_FILE=ACEE623F-B1A8-E411-8672-0025905938B4.root
1855
1856 mkdir -p /scratch/
1857 mount /dev/xvdf /scratch/
1858 cd /scratch
1859
1860 if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
1861     if [[ -e /root/.local/lib/aws/bin ]]; then
1862         export PATH=$PATH:/root/.local/lib/aws/bin
1863     else
1864         wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
1865         unzip awscli-bundle.zip
1866         python awscli-bundle/install
1867         export PATH=$PATH:/root/.local/lib/aws/bin
1868         #mkdir /root/.aws
1869         #mv /scratch/config /root/.aws/
1870         #chmod 400 /root/.aws/config
1871     fi
1872 fi
1873
1874 aws s3 cp --recursive s3://$TEST_BUCKET/s3_stresstest/ .
1875

```

```

1876 #Remember to keep the certificates valid and modify the script if the users number changes
... (The 0 is always for root)
1877 mv x509up_u50685 /tmp/x509up_u0
1878 chmod 400 /tmp/x509up_u0
1879 rm -f /etc/grid-security/certificates/*.r0
1880 voms-proxy-init -noregen -voms fermilab:/fermilab
1881
1882
1883 DEST=gsiftp://fndca1.fnal.gov:2811/fermigrid202/grassano
1884 #ID=$RANDOM
1885 echo "Upload test ID=$ID" > fermigrid_upload.log
1886 #wait $WAIT_ID
1887 echo "" > data.txt
1888 LIST="1 5 10 20"
1889 if [[ $# -gt 0 ]]; then
1890     LIST="$@"
1891 fi
1892 if [[ $# -eq 1 && $1 -eq 0 ]]; then
1893     kill $WAIT_ID
1894     exit
1895 fi
1896 for NUM in $LIST; do
1897     #mkdir -p $NUM
1898     echo "Simultaneous upload of $NUM copies..." >>fermigrid_upload.log
1899     echo "" > data.txt
1900     for n in `seq $NUM`; do
1901         echo "file:///scratch/$TEST_FILE $DEST/$ID/$n.root" >> data.txt
1902     done
1903     date >>fermigrid_upload.log
1904     if [[ $NUM -le 5 ]]; then
1905         CONCURRENCY=$NUM
1906     else
1907         CONCURRENCY=5
1908     fi
1909     globus-url-copy -cd -fast -p 4 -cc $CONCURRENCY -v -f data.txt >> command.log 2>&1
1910     date >>fermigrid_upload.log
1911     echo -e "edg-gridftp-ls -v $DEST/$ID/ | grep 1073604531 | wc -l` out of $NUM successfully
... uploaded\n" >>fermigrid_upload.log

```

```

1912     for n in `seq $NUM`; do
1913         edg-gridftp-rm $DEST/$ID/$n.root
1914     done
1915     edg-gridftp-rmdir $DEST/$ID/
1916     #let NUM*=10
1917     #echo $NUM
1918 done
1919
1920 exit
1921
1922 -----
1923 -----
1924 -----
1925
1926
1927 File: predictiondecisionengine/trunk/bin/Fermilab/run-test_fermigrid2.sh
1928 #!/bin/bash
1929
1930 TIME=$1
1931 shift
1932 ID=$1
1933 shift
1934 sleep $((180 + `date -d "$TIME" +%s` - `date +%s`)) &
1935 WAIT_ID=$!
1936
1937 TEST_BUCKET=grassano-test
1938 TEST_FILE=ACEE623F-B1A8-E411-8672-0025905938B4.root
1939
1940 mkdir -p /scratch/
1941 mount /dev/xvdf /scratch/ >/dev/null 2>&1
1942 cd /scratch
1943
1944 if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
1945     if [[ -e /root/.local/lib/aws/bin ]]; then
1946         export PATH=$PATH:/root/.local/lib/aws/bin
1947     else
1948         wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
1949         unzip awscli-bundle.zip

```

```

1950         python awscli-bundle/install
1951         export PATH=$PATH:/root/.local/lib/aws/bin
1952         #mkdir /root/.aws
1953         #mv /scratch/config /root/.aws/
1954         #chmod 400 /root/.aws/config
1955     fi
1956 fi
1957
1958 aws s3 cp --recursive s3://$TEST_BUCKET/s3_stresstest/ .
1959
1960 #Remember to keep the certificates valid and modify the script if the users number changes
1961 ... (The 0 is always for root)
1962 cp x509up_u2904 /tmp/x509up_u0
1963 chmod 400 /tmp/x509up_u0
1964 mv x509up_u2904 /tmp/x509up_u2904
1965 chmod 400 /tmp/x509up_u2904
1966 export X509_USER_CERT=/tmp/x509up_u2904
1967 export X509_USER_KEY=/tmp/x509up_u2904
1968 rm -f /etc/grid-security/certificates/*.r0
1969 voms-proxy-init -noregen -voms fermilab:/fermilab
1970
1971 DEST_XRD=root://cmseos.fnal.gov//eos/uscms/store/user/timm
1972 DEST_SRM=srm://cmseos.fnal.gov:8443/srm/v2/server?SFN=/eos/uscms/store/user/timm
1973 #ID=$RANDOM
1974 echo "Upload test ID=$ID" > fermigrid_upload.log
1975 LIST="1 5 10 20"
1976 if [[ $# -gt 0 ]]; then
1977     LIST="$@"
1978 fi
1979 if [[ $# -eq 1 && $1 -eq 0 ]]; then
1980     kill $WAIT_ID
1981     exit
1982 fi
1983 srmkdir $DEST_SRM/$ID
1984 wait $WAIT_ID
1985 for NUM in $LIST; do
1986     echo "Simultaneous upload of $NUM copies..." >>fermigrid_upload.log

```

```

1987     date >>fermigrid_upload.log
1988     for n in `seq $NUM`; do
1989         #srmcp -streams_num=5 file:/// $TEST_FILE $DEST_SRM/$ID/$n.root >> command.log 2>&1 &
1990         xrdcp -f -v -N -S 4 /scratch/$TEST_FILE $DEST_XRD/$ID/$n.root >> command.log 2>&1 &
1991     done
1992     wait
1993     date >>fermigrid_upload.log
1994
1995     echo -e "`srm ls $DEST_SRM/$ID/ | grep -c 1073604531` out of $NUM successfully uploaded\n"
1996 >>fermigrid_upload.log
1997     for n in `seq $NUM`; do
1998         srmrm $DEST_SRM/$ID/$n.root &
1999     done
2000     wait
2001 done
2002 srmrmdir $DEST_SRM/$ID/
2003
2004 exit
2005
2006 -----
2007 -----
2008
2009
2010 File: predictiondecisionengine/trunk/bin/Fermilab/run-tt_bar_gensim.sh
2011 #!/bin/bash
2012
2013 shift
2014 shift
2015
2016 mkdir -p /scratch/
2017 mount /dev/xvdf /scratch/
2018 cd /scratch
2019
2020 if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
2021     if [[ -e /root/.local/lib/aws/bin ]]; then
2022         export PATH=$PATH:/root/.local/lib/aws/bin
2023     else

```

```

2024         wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
2025         unzip awscli-bundle.zip
2026         python awscli-bundle/install
2027         export PATH=$PATH:/root/.local/lib/aws/bin
2028         #mkdir /root/.aws
2029         #mv /scratch/config /root/.aws/
2030         #chmod 400 /root/.aws/config
2031     fi
2032 fi
2033
2034 aws s3 cp --recursive s3://$TEST_BUCKET/tt_bar_gensim/ .
2035
2036
2037 tar -zxf benchmark-2015.tgz
2038
2039 groupadd cms_admin
2040 useradd -g cms_admin cms_admin
2041
2042 ln -s /scratch/cms/ /tmp/cms
2043 chown -R cms_admin.cms_admin /tmp/cms/
2044
2045 if [[ $# -eq 1 && $1 -eq 0 ]]; then
2046     exit
2047 fi
2048 cd /tmp/cms/
2049 su cms_admin -c "sh benchmark-2015-gensim.sh 1">log
2050 mv results results.Single_core
2051
2052 su cms_admin -c "sh benchmark-2015-gensim.sh">log
2053 mv results results.All_cores
2054
2055 -----
2056 -----
2057 -----
2058
2059
2060 File: predictiondecisionengine/trunk/bin/Fermilab/run-tt_bar_reco.sh
2061 #!/bin/bash

```

```

2062
2063 groupadd cms_admin
2064 useradd -g cms_admin cms_admin
2065
2066 mkdir -p /scratch/
2067 mount /dev/xvdf /scratch/
2068 cd /scratch
2069
2070 if [[ `which aws` = "" || `which aws | grep -F "no aws" != "" ]]; then
2071     if [[ -e /root/.local/lib/aws/bin ]]; then
2072         export PATH=$PATH:/root/.local/lib/aws/bin
2073     else
2074         wget https://s3.amazonaws.com/aws-cli/awscli-bundle.zip
2075         unzip awscli-bundle.zip
2076         python awscli-bundle/install
2077         export PATH=$PATH:/root/.local/lib/aws/bin
2078         #mkdir /root/.aws
2079         #mv /scratch/config /root/.aws/
2080         #chmod 400 /root/.aws/config
2081     fi
2082 fi
2083
2084 aws s3 cp s3://$TEST_BUCKET/tt_bar_reco/* ./
2085
2086 tar -zxf /scratch/benchmark-2015.tgz
2087 ln -s /scratch/tmp/cms/ /tmp/cms
2088
2089 mv /scratch/ACEE623F-B1A8-E411-8672-0025905938B4.root /tmp/cms
2090 mv /scratch/ttbarGENSIM.root /tmp/cms
2091 mv /scratch/step2-50ns-4x-6400.root /tmp/cms
2092 mv -f /scratch/*.sh /tmp/cms
2093
2094 mv /scratch/frontier-cache.tgz /tmp/cms
2095 cd /tmp/cms/
2096 tar -zxf /tmp/cms/frontier-cache.tgz
2097
2098 chown -R cms_admin.cms_admin /tmp/cms/
2099

```

```

2100 su cms_admin -c "sh start_x_benchmarks_reco.sh">>log &
2101
2102 exit
2103
2104 -----
2105 -----
2106 -----
2107
2108
2109 File: predictiondecisionengine/trunk/bin/Fermilab/runparallel.sh
2110 #!/bin/bash
2111
2112 cd ./hepspec2006/install
2113 mkdir ./results
2114 mv ./result ./results/Single_core
2115
2116 . ./shrc
2117 COUNT=`grep -c "^processor" /proc/cpuinfo`;
2118 for i in `seq $COUNT`; do
2119     runspec --config=linux64-gcc_cern.cfg all_cpp &
2120 done
2121 wait
2122
2123 mv ./result ./results/All_cores
2124
2125 -----
2126 -----
2127 -----
2128
2129
2130 File: predictiondecisionengine/trunk/bin/functions.sh
2131 #Description: Read the choice to a query and confront it with a list of correct answer
... limiting the numbers of checks
2132 #Parameters: 1st parameter: number of attempts possible other parameter:List of correct
... answers
2133 #Usage example (combined with make_list): make_list ... read_choice 3 `seq $LENGHT`
2134 function read_choice()
2135 {

```

```

2136     local NUM=$1
2137     shift
2138     CHOICES={"${@}" }
2139     while "true"; do
2140         read CHOICE
2141         CHOICE=`echo $CHOICE | tr YESNO yesno`
2142         for CMD in ${CHOICES[@]}; do
2143             if [[ "$CHOICE" = "$CMD" ]]; then
2144                 return 0
2145             fi
2146         done
2147         echo "Choice is out of range. Enter again : "
2148         let C_COUNT++
2149         if [[ $C_COUNT -ge $NUM ]]; then
2150             echo "$NUM invalid choices. Exiting..."
2151             exit
2152         fi
2153     done
2154 }
2155
2156 #Description: Make a numbered list out of an array or list of elements and return the number
2157 ... of elements in LENGHT
2158 #Parameters: array or list of elements
2159 #Usage example: make_list "$LIST" or make_list "${array[@]}"
2160 function make_list()
2161 {
2162     local C_COUNT=0
2163     if [[ `echo "$1" | wc -l` -gt 1 ]]; then
2164         while read -r line; do
2165             let C_COUNT++
2166             echo -e "$C_COUNT-\t$line"
2167         done <<< "$1"
2168     else
2169         local array={"${@}" }
2170         for CMD in ${array[@]}; do
2171             let C_COUNT++
2172             echo -e "$C_COUNT-\t$CMD"
2173         done

```

```

2173     fi
2174     LENGHT=$C_COUNT
2175 }
2176
2177 #Description: Transfor a lsit of elements written on different lines into an array
2178 #Parameters: list of element
2179 #Usage example: array=$(list_to_array "$LIST")
2180 function list_to_array()
2181 {
2182     #echo "$@"
2183     local COUNT=0
2184     local array
2185     while read -r line; do
2186         array[$COUNT]="$line"
2187         let COUNT++
2188     done <<< $@
2189     echo ${array[@]}
2190 }
2191
2192 -----
2193 -----
2194 -----
2195
2196
2197 File: predictiondecisionengine/trunk/bin/hepspec_crop_results.sh
2198 #!/bin/bash
2199
2200 B_PATH=/home/crivella/Fermilab/Benchmarks/hepspec06
2201 KEYS_PATH=/home/crivella/Fermilab
2202
2203 #Adjust this section with your own format used to name instances and keys#####
2204 VMS_DATA=`aws ec2 describe-instances --filters Name=tag-value,Values=grassano`
2205 VMS=`echo "$VMS_DATA" | grep -F "grassano" | cut -d "\"" -f 4`
2206 VMS_KEYS=`echo "$VMS_DATA" | grep -F "_grassano" | cut -d "\"" -f 4`
2207 VMS_PUB_DNS=`echo "$VMS_DATA" | grep -F "PublicDnsName" | cut -d "\"" -f 4 | uniq`
2208 #####
2209
2210

```

```

2211 : 'MANUAL_CHOICES
2212 NUM_VMS=0
2213 while read -r CMD; do
2214     let NUM_VMS++
2215     echo -e "$NUM_VMS-\t$CMD"
2216 done <<< "$VMS"
2217 echo -e "100-\tALL"
2218
2219 CHOICE=0
2220 echo -e "\nChoose VM to get results from: Or type -1 to exit\nFor multiple selescion separate
...
them with a space"
2221 read CHOICES
2222
2223 FLAG1=1
2224 while [ $FLAG1 -eq 1 ]; do
2225     FLAG1=0
2226     ALL_SELECT=0
2227     if [ $CHOICES -eq -1 ]; then
2228         echo "Quitting..."
2229         exit
2230     fi
2231     for CHOICE in $CHOICES; do
2232         if [ $CHOICE -gt $NUM_VMS ] || [ $CHOICE -lt 0 ] && [ $CHOICE -ne 100 ]; then
2233             FLAG1=1
2234             echo -e "Choice not available..."
2235         fi
2236         if [ $CHOICE -eq 100 ]; then
2237             ALL_SELECT=1
2238         fi
2239     done
2240     if [ $FLAG1 -eq 1 ]; then
2241         echo -e "\nChoose VM to get results from: Or type -1 to exit"
2242         echo "For multiple selescion separate them with a space"
2243         read CHOICES
2244     fi
2245 done
2246
2247 if [ $ALL_SELECT -eq 0 ]; then

```

```

2248     for CHOICE in $CHOICES; do
2249         let CHOICE=CHOICE-1
2250         VM=`echo "$VMS" | tail -n $(expr $NUM_VMS - $CHOICE) | head -n 1`
2251         VM_DNS=`echo "$VMS_PUB_DNS" | tail -n $(expr $NUM_VMS - $CHOICE) | head -n 1`
2252         VM_KEY=`echo "$VMS_KEYS" | tail -n $(expr $NUM_VMS - $CHOICE) | head -n 1`
2253
2254         KEY=`find $KEYS_PATH -name "$VM_KEY.pem"`
2255         if [ "$KEY" = "" ]; then
2256             echo "You do not have the necessary key. Terminating..."
2257             exit
2258         fi
2259
2260         mkdir $B_PATH/amazon/${VM:9:-8}
2261         scp -i $KEY -r root@$VM_DNS:/scratch/install/results/* $B_PATH/amazon/${VM:9:-8}/
2262     done
2263 else
2264     for CHOICE in `seq $NUM_VMS`; do
2265         let CHOICE=CHOICE-1
2266         VM=`echo "$VMS" | tail -n $(expr $NUM_VMS - $CHOICE) | head -n 1`
2267         VM_DNS=`echo "$VMS_PUB_DNS" | tail -n $(expr $NUM_VMS - $CHOICE) | head -n 1`
2268         VM_KEY=`echo "$VMS_KEYS" | tail -n $(expr $NUM_VMS - $CHOICE) | head -n 1`
2269         KEY=`find $KEYS_PATH -name "$VM_KEY.pem"`
2270         if [ "$KEY" = "" ]; then
2271             echo "You do not have the necessary key. Terminating..."
2272             exit
2273         fi
2274
2275         mkdir $B_PATH/amazon/${VM:9:-8}
2276         scp -i $KEY -r root@$VM_DNS:/scratch/install/results/* $B_PATH/amazon/${VM:9:-8}/
2277     done
2278 fi
2279
2280 MANUAL_CHOICES'
2281
2282 #AUTO_CHOICES
2283
2284 echo "AutoChoices"
2285 #Amazon cropping

```

```

2286 CHOICES=""
2287 NUM_VMS=0
2288
2289 while read -r CMD; do
2290     let NUM_VMS++
2291     #echo -e "$NUM_VMS-\t$CMD"
2292     if [[ `echo "$CMD" | grep -F "-hepspec" ` != "" ]]; then
2293         CHOICES="$CHOICES $NUM_VMS"
2294     fi
2295 done <<< "$VMS"
2296
2297 for CHOICE in $CHOICES; do
2298     VM=`echo "$VMS" | head -n $CHOICE | tail -n 1`
2299     VM_DNS=`echo "$VMS_PUB_DNS" | head -n $CHOICE | tail -n 1`
2300     VM_KEY=`echo "$VMS_KEYS" | head -n $CHOICE | tail -n 1`
2301
2302     KEY=`find $KEYS_PATH -name "$VM_KEY.pem"`
2303     if [ "$KEY" = "" ]; then
2304         echo "You do not have the necessary key for $VM. Terminating..."
2305         exit
2306     fi
2307
2308     mkdir $B_PATH/amazon/${VM:9:-12}
2309     scp -i $KEY -r root@$VM_DNS:/scratch/install/results/* $B_PATH/amazon/${VM:9:-12}/
2310 done
2311
2312 : '
2313 CHOICES="fermicloud148"
2314
2315 for CHOICE in $CHOICES; do
2316     mkdir $B_PATH/bare_metal/$CHOICE
2317     scp -r cms_admin@$CHOICE:./hepspec2006/install/results/* $B_PATH/bare_metal/$CHOICE/
2318 done
2319 '
2320
2321 RESULTS=`find $B_PATH -name "lock.CPU2006" | sort`
2322 while read -r CMD; do
2323     RES_PATH=`dirname $CMD`

```

```

2324     N_CORE=`find $RES_PATH -name "CPU2006*.log" | wc -l`
2325     for COUNT in `seq $N_CORE`; do
2326         if [ $COUNT -eq 1 ]; then
2327             echo "CORE$COUNT" > $RES_PATH/crop_core.txt
2328         else
2329             echo -e "\nCORE$COUNT" >> $RES_PATH/crop_core.txt
2330         fi
2331         cat $RES_PATH/CFP2006.`printf %03d $COUNT`.ref.txt | grep -F "*" | sort | uniq >>
... $RES_PATH/crop_core.txt
2332         cat $RES_PATH/CINT2006.`printf %03d $COUNT`.ref.txt | grep -F "*" | sort | uniq >>
... $RES_PATH/crop_core.txt
2333     done
2334 done <<< "$RESULTS"
2335
2336 -----
2337 -----
2338 -----
2339
2340
2341 File: predictiondecisionengine/trunk/bin/make_diff_sum.sh
2342 #!/bin/bash
2343 . /home/crivella/bin/functions.sh
2344
2345 RES_PATH=/home/crivella/Fermilab/Benchmarks/s3_stresstest_u/$1
2346 FILE="diff_sum_`date +%F`.log"
2347 RESULTS=`find $RES_PATH -name "diff.log" | sort -n`
2348 while read -r line; do
2349     echo "$line" >> $RES_PATH/$FILE
2350     LIST=`cat $line | grep "Diff for" | cut -d " " -f 1 | cut -d " " -f 3 | sort -n | uniq`
2351     for CMD in $LIST; do
2352         echo -ne "Sum of $CMD simultaneous transfer throughput: \t">> $RES_PATH/$FILE
2353         LINES=`cat $line | grep "Diff for $CMD is"`
2354         INTEGER=0
2355         DECIMAL=0
2356         while read -r line2; do
2357             VAR=`echo "$line2" | cut -d " " -f 5 | cut -d " " -f 1`
2358             INT=`echo $VAR | cut -d "." -f 1`
2359             if [[ -z "$INT" ]]; then

```

```

2360         INT=0
2361         fi
2362         INTEGER=`echo $INTEGER+$INT | bc -l`
2363         FLOAT=`echo $VAR | cut -d "." -f 2`
2364         DECIMAL=`echo $DECIMAL+$FLOAT | bc -l`
2365         #echo "$INT.$FLOAT -> $INTEGER.$DECIMAL"
2366     done <<< "$LINES"
2367     #echo #####
2368     while [[ $DECIMAL -ge 1000 ]]; do
2369         let DECIMAL-=1000
2370         let INTEGER+=1
2371     done
2372     echo "$INTEGER.`printf %03d $DECIMAL` GB/s">> $RES_PATH/$FILE
2373     done
2374     echo -e "\n" >> $RES_PATH/$FILE
2375 done <<< "$RESULTS"
2376 exit
2377
2378 -----
2379 -----
2380 -----
2381
2382
2383 File: predictiondecisionengine/trunk/bin/README_aws_launch_benchmark
2384 README for the aws_launch_benchmark.sh script
2385
2386 #####
2387 ... #####
2388 1. Index
2389     1- Index
2390     2- Required packages
2391     3- Useful variables
2392     4- Optional parameters
2393     5- Priorities
2394     6- Resume mode
2395     7- Adding new benchmarks to the script
2396 #####

```

```

2396... #####
2397 2. Required packages
2398     In order to work, this script will check for the presence of the aws and nmap packages. If
2399     one of them is missing, the script will
2400     terminate, asking the user to install them before proceeding.
2401 #####
2402 ... #####
2403 3. Useful variables
2404     FILES_PATH Path for the directory containing the run-benchmark_name.sh files
2405     KEYS_PATH Path for the directory containing the aws keys (.pem)
2406     TMP_PATH Path for the directory to store the tmp files
2407
2408     benchmark Name of the chosen benchmark to execute
2409     vol_size The size of the extra-volumes to create in order to make the specified
2410     benchmark work
2411     ROLE Role to attach at launch to the instances
2412     PARAM Additional parameters to pass to the run-$benchmark.sh script
2413     AMI_ID ID of the AMI to use for the instances
2414     SG_ID ID of the Security group to use for the instances
2415     KEY Name of the KEY (without .pem) to use for the instances
2416
2417     benchmarks Array containing all the benchmark that the script can run (The indexes need
2418     to be matching with those of the
2419     next arrays)
2420     req_vol_size Array containing the sizes of the extra-volumes for each benchmarks
2421     roles Array containing the role that the instances will need to execute the
2422     benchmark. If a value is null (ex: (" " "..."))
2423     then the default "AllowS3_Download" will be selected (required to download
2424     benchmark files from S3)
2425     parameters Array containing the default parameters to use with each benchmark. They can
2426     be overwritten at launch using the -p
2427     optional parameter (See more in paragraph 5)
2428
2429 #####
2430 ... #####
2431 4. Optional parameters
2432     -a Specify the ID of the AMI the script will try to use

```

```

2426     -s Specify the ID of the Security Group the script will try to use
2427     -k Specify the Name of the Key Pair the script will try to use
2428     -i Specify the kind and amount of instances to benchmark
2429         Example: aws_launch_benchmark.sh -i "3:c3.2xlarge 10:t2.micro 1:m3.4xlarge"
2430         This command will use the later specified benchmark on 3 c3.2xlarge instances, 10
... t2.micro and 1 m3.4xlarge
2431     -p Set optional parameters to pass to the "run-name_of_benchmark.sh" executed on the VM
2432         Example: aws_launch_benchmark.sh -p "1 10 100"
2433         For bandwidth benchmarks, this will make them do 1 simultaneous up/download of
... files, followed by 10 simultaneous and
2434             than 100 simultaneous
2435     -z Specify a profile to use, so to operate in different zone than the default one
2436         The name of the profile should be the same as the zone they are meant for
2437     -r Enable the resume mode (See paragraph 6 for more instruction)
2438
2439 #####
... #####
2440 5. Priorities
2441     To specify aws related parameters (AMI ID, KEY, SG_ID, Param, ZONE) there are 3 level of
... priorities in this script.
2442     The default values will be the ones specified in the script and will be used unless
... optional parameteres are being used. In this case
2443     they will override the default values.
2444     The highest level of priorities is based on a check with aws. If the values in use are not
... found on aws, the script will prompt the
2445     user and ask if he wants to see a list of the availables option on aws to choose from, or
... exit
2446
2447
2448 #####
... #####
2449 6. Resume mode
2450     The resume mode works only if launching one kind of instances (example: only c3.2xlarge or
... only m3.medium ...).
2451     Also, the benchmark the user is launching has to be the same that was previously launched
... on the instances he is trying to resume.
2452     Any other use could causes bug to arise
2453     When the resume mode is enabled, the script will look for a tmp file in the folder

```

```

2453... $TMP_PATH/benchmark_name and read it if found or
2454     exit otherwise.
2455     This file is generated automatically by a previous run of the script, but can be manually
... edited if needed
2456     The format of the file has to be
2457         NUM_VMS:INSTANCE_TYPE:INSTANCE_ID:PUB_DNS:KEY      (NUM_VMS start from 0)
2458     example:
... 0:m3.medium:i-d3ea6708:ec2-52-27-10-197.us-west-2.compute.amazonaws.com:mykey
2459         1:m3.medium:i-d4ea670f:ec2-52-88-72-102.us-west-2.compute.amazonaws.com:mykey
2460     The data present in the file will be stored in the same arrays used by the script to store
... the data of newly created instances, and
2461     will act as the starting point of the other parts of the script depending on the selected
... choice that the user will be prompted to take while in resume mode.
2462     This choice can be either to resume the script from "Volume creation" or from "Start
... Benchmark":
2463     Volume creation After creating the missing instances, the script will create a
... new volume for all instances, included
2464         the one in the tmp file. This function is meant to be used, if the script
... for some reason got
2465         interrupted before the extra-volume was created.
2466     Start Benchmark After creating the missing instances, the script will create a
... new volume only for the newly started
2467     instances. This is meant for launching 2 successive benchmark, where one
... is the extended version of
2468     the previous.
2469     For example. After launching a benchmark on 5 c3.2xlarge instances, the
... user wants to launch the same
2470     benchmark, but on 10 c3.2xlarges instances. The normal way would be to
... reuse the 5 instances
2471     previously launcher, and this can be accomplished with the resume mode
2472
2473 #####
... #####
2474 7. Adding new benchmarks to the script
2475     1- Modify the benchmarks, req_vol_size, roles, parameters array in the script adding the
... name of the benchmark in benchmarks,
2476     the size of the extra-volume needed for it to run in req_vol_size, th name of the role
... the benchmark will use in roles (use ""

```

```

2477         to set it to the default one), and the default parameter for that benchmark in
... parameters.
2478         Using 0 in req_vol_size will prompt a dynamic allocation of space based on the
... parameters. For example while executing a
2479         bandwidth download test, the req_vol_size will depend on the size of the files and
... their number. The script consider the
2480         default size for the files of 1GB, and will thus use the sum of the parameters+3 as
... the vol_size
2481         2- Load all the required files for your benchmark on s3
2482         3- Modify the section of the script that checks if you have an s3 folder in you bucket
... for the benchmark in case you don't want
2483         it to have the same name as the benchmark (example: different benchmarks share the
... same files)
2484         4- Create a run-benchmark_name.sh script and put it in the directory conciding with the
... var FILES_PATH
2485         This script will be executed as root from the /root/ directory and will need to make
... the file sys for the extra-disk and mount
2486         it. It will need afterwards to download the required files from s3 and place them
... where needed, and afterwards run all the
2487         commands needed to execute the benchmarks.
2488         See the already writte run-*.sh files for examples
2489         Note: The first parameters passed from the aws_launch_benchmark.sh script to the
... run-*.sh will always be the time right before going
2490         into the Start Benchmark phase and the number identifying the job. The first is used
... to synchronize bandwidth benchmarks.
2491         If the run-*.sh has other parameters, the user will need to take this into account.
... Using the "shift" command will shift the
2492         parameter indexes from n to n-1, deleting the $1. This can be a useful way to ignore
... the the first 2 default parameters,
2493         enabling a easyier use of the @$ to invoke the list of user defined parameters
2494
2495
2496 -----
2497 -----
2498 -----
2499
2500
2501 File: predictiondecisionengine/trunk/bin/s3_manager.sh

```

```

2502 #!/bin/bash
2503
2504 #D_PATH=$HOME/s3_download
2505 #if [[ ! -d $D_PATH ]]; then
2506 #   echo "The Download directory doesn't exist. Do you wish to create it on the default
... path?(y/n)"
2507 #else
2508 #   CHOICE="bypass"
2509 #fi
2510
2511 A_PATH=`find $HOME -name "s3_download_path" 2>/dev/null`
2512 D_PATH=`dirname $A_PATH`
2513 if [[ "$D_PATH" != "" ]]; then
2514     echo "Download path found as $D_PATH"
2515     CHOICE="bypass"
2516 else
2517     echo "The Download directory doesn't exist. Do you wish to create it on the default
... path?(y/n)"
2518     D_PATH=$HOME/s3_download
2519 fi
2520
2521
2522 COUNT=0
2523 while [[ "$CHOICE" != "y" && "$CHOICE" != "n" && "$CHOICE" != "bypass" ]]; do
2524     read CHOICE
2525     if [[ "$CHOICE" = "y" ]]; then
2526         if mkdir -p $D_PATH; then
2527             :
2528         else
2529             echo "Cannot create the directory. Exiting..."
2530             exit
2531         fi
2532     elif [[ "$CHOICE" = "n" ]]; then
2533         echo "Do you wish to specify a different path (y) or exit (n)?"
2534         read CHOICE2
2535         if [[ "$CHOICE2" = "y" ]]; then
2536             COUNT2=0
2537             CHECK=0

```

```

2538         while [[ $CHECK -eq 0 ]]; do
2539             echo "Insert the path: "
2540             read D_PATH
2541             if mkdir -p $D_PATH; then
2542                 CHECK=1
2543             else
2544                 let COUNT2++
2545                 echo "Can't create the folder(try number $COUNT2). Enter path again"
2546             fi
2547             if [[ $COUNT2 -ge 3 ]]; then
2548                 echo "3 invalid choices. Exiting..."
2549             fi
2550         done
2551
2552         else
2553             exit
2554         fi
2555     else
2556         let COUNT++
2557         echo "Choise is not valid. Enter it again"
2558     fi
2559     if [[ $COUNT -ge 3 ]]; then
2560         echo "3 invalid choices. Exiting..."
2561         exit
2562     fi
2563
2564 done
2565 if [[ -d $D_PATH ]]; then
2566     echo "$D_PATH" > $D_PATH/s3_download_path
2567 fi
2568
2569 echo -e "Select the operaion to perform:\n1- Download files\n2- Upload files\n10-exit"
2570 read CHOICE
2571 if [[ $CHOICE -eq 10 ]]; then
2572     exit
2573 fi
2574
2575 USER=grassano

```

```

2576 BUCKETS_LIST=`aws s3api list-buckets | grep Name | cut -d "\"" -f 4`
2577 COUNT=0
2578 BUCKETS[0]=`echo "$BUCKETS_LIST" | wc -l`
2579 while read -r CMD; do
2580     let COUNT++
2581     BUCKETS[$COUNT]=$CMD
2582     echo -e "$COUNT-\t$CMD"
2583 done <<< "$BUCKETS_LIST"
2584 echo "Choose the bucket to perform the action on:"
2585 CHOICE2=0
2586 COUNT=0
2587 while [[ $CHOICE2 -lt 1 || $CHOICE2 -gt ${BUCKETS[0]} ]]; do
2588     let COUNT++
2589     if [[ $COUNT -ge 4 ]]; then
2590         echo "3 invalid choices. Exiting..."
2591     fi
2592     read CHOICE2
2593     if [[ $CHOICE2 -lt 1 || $CHOICE2 -gt ${BUCKETS[0]} ]]; then
2594         echo "Choice out of range. Enter again:"
2595     fi
2596 done
2597
2598 if [[ $CHOICE -eq 1 ]]; then
2599     OBJ_DATA=`aws s3api list-objects --bucket ${BUCKETS[$CHOICE2]}`
2600     OBJ_LIST=`echo "$OBJ_DATA" | grep Key | cut -d "\"" -f 4`
2601     COUNT=0
2602     OBJ[0]=`echo "$OBJ_LIST" | wc -l`
2603     while read -r CMD; do
2604         let COUNT++
2605         OBJ[$COUNT]=$CMD
2606         echo -e "$COUNT-\t$CMD"
2607     done <<< "$OBJ_LIST"
2608     echo "Choose the objects you want to download (num1 num2 ...)"
2609     read DOWNLOADS
2610     for CMD in $DOWNLOADS; do
2611         if [[ $CMD -ge 1 && $CMD -le ${OBJ[0]} ]]; then
2612             echo "Downloading ${OBJ[$CMD]}..."
2613             aws s3 cp s3://${BUCKETS[$CHOICE2]}/${OBJ[$CMD]} $D_PATH

```

```
2614         else
2615             echo "$CMD is an invalid selection. Ignoring it..."
2616         fi
2617     done
2618 elif [[ $CHOICE -eq 2 ]]; then
2619     if [[ $# -gt 0 ]]; then
2620         echo "Taking parameter from function input"
2621         UPLOADS=$@
2622     else
2623         echo "Insert files to upload (name1 name2 ...)"
2624         read UPLOADS
2625     fi
2626     for CMD in $UPLOADS; do
2627         if [[ -f $CMD ]]; then
2628             echo "Uploading $CMD"
2629             aws s3 cp $CMD s3://${BUCKETS[$CHOICE2]}/
2630         else
2631             echo "$CMD is not a regular file. Ignoring it..."
2632         fi
2633     done
2634 fi
2635
2636 exit
2637
2638 -----
2639 -----
2640 -----
2641
2642
2643
```

```

1 | File: monitoringaccountingbilling/trunk/src/gratia/awsvm/aws-gratia-probe-fix
2 | #!/usr/bin/env python
3 | import gratia.common.Gratia as Gratia
4 | import gratia.common.GratiaCore as GratiaCore
5 | import gratia.common.GratiaWrapper as GratiaWrapper
6 | from gratia.common.Gratia import DebugPrint
7 | import boto3;
8 | from pprint import pprint;
9 | import datetime
10 | from time import mktime
11 | import time
12 | import sys
13 | from gratia.awsvm.spot_price import spot_price
14 | from gratia.awsvm.cpuutil import cpuUtilization
15 | from gratia.awsvm.inst_hardware import insthardware
16 | from gratia.awsvm.get_account_id import get_account_id
17 | import os
18 |
19 | class Awsgratiaprobe:
20 |     def __init__(self):
21 |         GratiaCore.Config.set_DebugLevel(5)
22 |
23 |
24 |     def process(self):
25 |         ec2=boto3.client('ec2',region_name='us-west-2')
26 |         mygacid=get_account_id()
27 |         owneracct=mygacid.get_id()
28 |         response = ec2.describe_instances()
29 |         #pprint(response)
30 |         resv=response['Reservations']
31 |         for reservation in resv:
32 |             #pprint(reservation)
33 |             instances=reservation['Instances']
34 |             for instance in instances:
35 |                 #pprint(instance)
36 |                 print instance['InstanceId']
37 |                 #print instance['InstanceType']
38 |                 r = Gratia.UsageRecord()

```

```

39 |         user="aws account user"
40 |         project="aws-no project name given"
41 |         voname="fermilab"
42 |         try:
43 |             tags=instance['Tags']
44 |             print "the tags are"
45 |             for tag in tags:
46 |                 print tag['Key'],
47 |                 print tag['Value']
48 |             for tag in tags:
49 |                 if tag['Key'].lower() == 'user'.lower():
50 |                     user=tag['Value']
51 |             for tag in tags:
52 |                 if tag['Key'].lower() == 'name'.lower():
53 |                     r.JobName(tag['Value'])
54 |             for tag in tags:
55 |                 if tag['Key'].lower() == 'project'.lower():
56 |                     project=tag['Value']
57 |             #         voname=tag['Value']
58 |             for tag in tags:
59 |                 if tag['Key'].lower() == 'vo'.lower() or tag['Key'].lower() ==
... 'voname'.lower():
60 |                     voname=tag['Value']
61 |         except KeyError:
62 |             print 'no tags'
63 |         r.LocalUserId(user)
64 |         r.GlobalUserId(owneracct)
65 |         r.ProjectName(project)
66 |         r.VOName(voname)
67 |         r.ReportableVOName(voname)
68 |         #Public Ip address is retrieved if instance is running"
69 |         try:
70 |             ipaddr=instance['PublicIpAddress']
71 |             r.MachineName(instance['PublicIpAddress'],instance['ImageId'])
72 |         except KeyError:
73 |             r.MachineName("no Public ip as instance has been stopped",instance['ImageId'])
74 |
75 |

```

```

76         r.LocalJobId(instance['InstanceId'])
77         r.GlobalJobId(instance['InstanceId']+"#+"+repr(time.time()))
78     #print 'hello1'
79     try:
80         for tag in tags:
81             if tag['Key'].lower() == 'name'.lower():
82                 r.JobName(tag['Value'])
83     except KeyError:
84         print 'no tags'
85     #status,description=recs[i].getStatus()
86
87     state=instance['State']
88     #print state['Name']
89     status=1
90     if state['Name']=="running":
91         print status
92         status=0
93     else:
94         status=1
95     #print 'hello'
96     #print status
97     pprint(r)
98     #print instance['StateTransitionReason']
99     description=instance['StateTransitionReason']
100    #print description
101    r.Status(status,description)
102
103    #
104    r.ProcessorsDescription(instance['InstanceType'])
105    #r.MachineNameDescription(instance['ImageId'])
106
107    try:
108        ipaddr=instance['PrivateIpAddress']
109
110    r.Host(instance['PrivateIpAddress'],False,instance['Placement']['AvailabilityZone'])
111    r.SubmitHost(instance['PrivateIpAddress'],instance['Placement']['AvailabilityZone'])
112    except KeyError:
113        r.SubmitHost("no Private ip as instance has been terminated")

```

```

112    #print 'setting site name'
113
114    #GratiaCore.Config.setSiteName('aws'+instance['Placement']['AvailabilityZone'])
115    #print 'done setting'
116    #r.ReportedSiteName('aws'+instance['Placement']['AvailabilityZone'])
117    r.ResourceType('aws')
118    r.Njobs(1,"The no of jobs running at a time")
119    r.NodeCount(1) # default to total
120
121    try:
122        hardwdet=GratiaCore.Config.getConfigAttribute("HardwareDetails")
123    except Exception as e:
124        print "The File for hardware details is not present. Pls check config file
... Hardware details value."
125    else:
126        hardwdet="/usr/share/gratia/awsvm/hardwareinst"
127        instdet=insthardware(hardwdet)
128        types=instdet.gettypedetails()
129        pprint(types)
130        processor='1'
131        memory=''
132        price=0.0
133        for t in types:
134            print t['instance-type'], instance['InstanceType']
135            if t['instance-type'] == instance['InstanceType']:
136                pprint (t)
137                processor=t['vcpu']
138                memory=t['ram']
139                price=t['price']
140                print processor,memory,price
141            #print memory," the value of memory"
142            cpu=float(processor)
143            print cpu
144            r.Processors(int(cpu),0,"total",instance['InstanceType'])
145            r.Memory(float(memory))
146            chargedesc=""
147            # Spot price is retrieved using instance id as the charge per hour of that
... instance in the last hour
148            #print instance['InstanceId']

```

```

148         if "SpotInstanceRequestId" in instance.keys():
149             sp=spot_price.spot_price()
150             value=sp.get_price(instance['InstanceId'])
151             #print value
152             price=value
153             chargedesc="The instance is a on-demand instance hence charge is fixed per
... hour"
154         if status==1:
155             price=0
156             chargedesc="The spot price charged in last hour corresponding to launch time"
157             r.Charge(str(price),"$","$/instance hr",chargedesc)
158             # The Time period for which the spot price and other values are calculated is noted
... down
159             launchtime=instance['LaunchTime']
160             #print launchtime.hour
161             #print 'hello3'
162             minu=launchtime.minute
163             #print minu
164
165             currtime=time.time()
166
167
168             EndTime=datetime.datetime.now()
169             #print type(EndTime)
170             EndTime=EndTime.replace(minute=minu)
171             StartTime=EndTime.replace(hour=(EndTime.hour-1))
172             #print StartTime
173             #print EndTime
174             #print 'starttime'
175             t=StartTime.date()
176             #print GratiaCore.TimeToString(time.mktime(t.timetuple()))
177             #print 'convert'
178             stime=time.mktime(StartTime.timetuple())
179             r.StartTime(stime)
180
181             et=EndTime.date()
182             etime=time.mktime(EndTime.timetuple())
183             r.EndTime(etime)

```

```

184         #print 'hello123'
185         #print etime-stime," the diff"
186         r.WallDuration(etime-stime)
187         Cpu=cpuUtilization()
188         aver=Cpu.getUtilPercent(instance['InstanceId'])
189         #print aver," average in percentage"
190         #print type(aver)
191         #print type(cpu)
192         if aver is None:
193             cpuUtil=0.0
194             print "The CPU Utilization value is NULL as the instance was not running in the
... last hour"
195             r.CpuDuration(0,'user')
196         else:
197             cpuUtil=aver
198             r.CpuDuration((etime-stime)*float(aver)*cpu/100,'user')
199             r.CpuDuration(0,'system')
200             r.ResourceType("AWSVM")
201             r.AdditionalInfo("Version","1.0")
202
203             #print r
204             print 'done'
205             Gratia.Send(r)
206
207
208
209
210 if __name__ == '__main__':
211     try:
212         Gratia.Initialize('/etc/gratia/awsvm/ProbeConfig')
213         GratiaWrapper.CheckPreconditions()
214         vmProbe=Awsgratiaprobe()
215         Filelock="filelock"
216         conf=GratiaCore.Config
217         duplicatelock=conf.getConfigAttribute("ExemptDuplicates")
218         filelock=conf.getConfigAttribute("DuplicateFilelock")
219         if duplicatelock == "True":
220             if os.path.isfile(Filelock):

```

```

221         fl=open(Filelock, 'r+')
222         date=datetime.datetime.now()
223         line=fl.readline()
224         print line
225         prevdate = datetime.datetime.strptime(line, "%Y-%m-%d %H:%M:%S.%f")
226         print prevdate
227         currtime=time.mktime(date.timetuple())
228         prevtime=time.mktime(prevdate.timetuple())
229         diff=currtime-prevtime
230         print diff
231         if diff>=3599.0:
232             fl.seek(0, 0)
233             fl.truncate()
234             fl.write(str(date));
235             t = os.path.getmtime(Filelock)
236             print t
237             print datetime.datetime.fromtimestamp(t)
238             fl.close()
239         vmProbe.process()
240     else:
241         print "hour is not over yet"
242         fl.close()
243     else:
244         fl=open(Filelock, 'w+')
245         date=str(datetime.datetime.now())
246         fl.write(date);
247         fl.close()
248         vmProbe.process()
249     else:
250         vmProbe.process()
251 except Exception, e:
252     print >> sys.stderr, str(e)
253     sys.exit(1)
254 sys.exit(0)
255
256 -----
257 -----
258 -----

```

```

259 -----
260
261
262 File: monitoringaccountingbilling/trunk/src/gratia/awsvm/aws-gratia-probe-multi-fix
263 #!/usr/bin/env python
264 import gratia.common.Gratia as Gratia
265 import gratia.common.GratiaCore as GratiaCore
266 import gratia.common.GratiaWrapper as GratiaWrapper
267 from gratia.common.Gratia import DebugPrint
268 import boto3;
269 from boto3.session import Session
270 from pprint import pprint;
271 import datetime
272 from time import mktime
273 import time
274 import sys
275 from gratia.awsvm.spot_price import spot_price
276 from gratia.awsvm.cputil import cpuUtilization
277 from gratia.awsvm.inst_hardware import insthardware
278 from gratia.awsvm.get_account_id import get_account_id
279 import os
280
281 class Awsgratiaprobe:
282     def __init__(self):
283         GratiaCore.Config.set_DebugLevel(5)
284
285
286     def process(self):
287         for account in ( 'rnd', 'cms', 'nova', 'fermilab' ):
288             session = Session(profile_name = account)
289             for region in ( 'us-west-2', 'us-west-1', 'us-east-1' ):
290                 ec2=boto3.client('ec2', region_name=region)
291                 mygacid=get_account_id()
292                 owneracct=mygacid.get_id()
293                 print owneracct
294                 response = ec2.describe_instances()
295                 #pprint(response)
296                 resv=response['Reservations']

```

```

297         for reservation in resv:
298             #pprint(reservation)
299             instances=reservation['Instances']
300             for instance in instances:
301                 #pprint(instance)
302                 print instance['InstanceId']
303                 #print instance['InstanceType']
304                 r = Gratia.UsageRecord()
305                 # set the defaults
306                 user="aws account user"
307                 project="aws-no project name given"
308                 voname="fermilab"
309                 try:
310                     tags=instance['Tags']
311                     print "the tags are"
312                     for tag in tags:
313                         print tag['Key'],
314                         print tag['Value']
315                     for tag in tags:
316                         if tag['Key'].lower() == 'user'.lower():
317                             user=tag['Value']
318                     for tag in tags:
319                         if tag['Key'].lower() == 'name'.lower():
320                             r.JobName(tag['Value'])
321                     for tag in tags:
322                         if tag['Key'].lower() == 'project'.lower():
323                             project=tag['Value']
324                     for tag in tags:
325                         if tag['Key'].lower() == 'vo'.lower() or tag['Key'].lower() ==
... 'voname'.lower():
326                             voname=tag['Value']
327                 except KeyError:
328                     print 'no tags'
329                 r.LocalUserId(user)
330                 r.GlobalUsername(owneracct)
331                 r.ProjectName(project)
332                 r.VOName(voname)
333                 r.ReportableVOName(voname)

```

```

334                 #Public Ip address is retrieved if instance is running"
335             try:
336                 ipaddr=instance['PublicIpAddress']
337                 r.MachineName(instance['PublicIpAddress'],instance['ImageId'])
338             except KeyError:
339                 r.MachineName("no Public ip as instance has been
... stopped",instance['ImageId'])
340
341
342                 r.LocalJobId(instance['InstanceId'])
343                 r.GlobalJobId(instance['InstanceId']+"#"+repr(time.time()))
344             #print 'hello1'
345             try:
346                 for tag in tags:
347                     if tag['Key'].lower() == 'name'.lower():
348                         r.JobName(tag['Value'])
349             except KeyError:
350                 print 'no tags'
351             #status,description=reecs[i].getStatus()
352
353             state=instance['State']
354             #print state['Name']
355             #set the default status
356             status=1
357             if state['Name']=="running":
358                 print status
359                 status=0
360             else:
361                 status=1
362             #print 'hello'
363             #print status
364             pprint(r)
365             #print instance['StateTransitionReason']
366             description=instance['StateTransitionReason']
367             #print description
368             r.Status(status,description)
369
370             #
371             r.ProcessorsDescription(instance['InstanceType'])

```

```

371         #r.MachineNameDescription(instance['ImageId'])
372
373         try:
374             ipaddr=instance['PrivateIpAddress']
375
... r.Host(instance['PrivateIpAddress'],False,instance['Placement']['AvailabilityZone'])
376
... r.SubmitHost(instance['PrivateIpAddress'],instance['Placement']['AvailabilityZone'])
377         except KeyError:
378             r.SubmitHost("no Private ip as instance has been terminated")
379         #print 'setting site name'
380
381
... #GratiaCore.Config.setSiteName('aws'+instance['Placement']['AvailabilityZone'])
382         #print 'done setting'
383         #r.ReportedSiteName('aws'+instance['Placement']['AvailabilityZone'])
384         r.ResourceType('aws')
385         r.Njobs(1,"The no of jobs running at a time")
386         r.NodeCount(1) # default to total
387         try:
388             hardwdet=GratiaCore.Config.getConfigAttribute("HardwareDetails")
389         except Exception as e:
390             print "The File for hardware details is not present. Pls check config
... file Hardware details value."
391         else:
392             hardwdet="/usr/share/gratia/awsvm/hardwareinst"
393             instdet=insthardware(hardwdet)
394             types=instdet.gettypedetails()
395         #
396             pprint(types)
397             processor='1'
398             memory=''
399             price=0.0
400         for t in types:
401             #
402                 print t['instance-type'], instance['InstanceType']
403                 if t['instance-type'] == instance['InstanceType']:
404                     #
405                         pprint (t)
406                         processor=t['vcpu']
407                         memory=t['ram']

```

```

405             price=t['price']
406         #
407                 print processor,memory,price
408                 #print memory," the value of memory"
409                 cpu=float(processor)
410                 #
411                 print cpu
412                 r.Processors(int(cpu),0,"total",instance['InstanceType'])
413                 r.Memory(float(memory))
414                 chargedesc=""
415                 # Spot price is retrieved using instance id as the charge
... per hour of that instance in the last hour
416                 #print instance['InstanceId']
417                 if "SpotInstanceRequestId" in instance.keys():
418                     sp=spot_price.spot_price()
419                     value=sp.get_price(instance['InstanceId'])
420                     #print value
421                     price=value
422                     chargedesc="The instance is a on-demand instance hence charge is fixed
... per hour"
423                 if status==1:
424                     price=0
425                     chargedesc="The spot price charged in last hour corresponding to launch
... time"
426                 r.Charge(str(price),"$/instance hr",chargedesc)
427                 # The Time period for which the spot price and other values are calculated
... is noted down
428                 launchtime=instance['LaunchTime']
429                 #print launchtime.hour
430                 minu=launchtime.minute
431                 #print minu
432
433                 currtime=time.time()
434
435                 EndTime=datetime.datetime.now()
436                 EndTime=EndTime.replace(minute=minu)
437                 StartTime=EndTime.replace(hour=(EndTime.hour-1))
438                 #print 'starttime'
439                 t=StartTime.date()

```

```

439         #print GratiaCore.TimeToString(time.mktime(t.timetuple()))
440         #print 'convert'
441         stime=time.mktime(StartTime.timetuple())
442         r.StartTime(stime)
443
444         et=EndTime.date()
445         etime=time.mktime(EndTime.timetuple())
446         r.EndTime(etime)
447         r.WallDuration(etime-stime)
448         Cpu=cpuUtilization()
449         aver=Cpu.getUtilPercent(instance['InstanceId'])
450         #print aver," average in percentage"
451         #print type(aver)
452         #print type(cpu)
453         if aver is None:
454             cpuUtil=0.0
455             print "The CPU Utilization value is NULL as the instance was not
... running in the last hour"
456             r.CpuDuration(0,'user')
457         else:
458             cpuUtil=aver
459             r.CpuDuration((etime-stime)*float(aver)*cpu/100,'user')
460             r.CpuDuration(0,'system')
461             r.ResourceType("AWSVM")
462             r.AdditionalInfo("Version","1.0")
463
464         #print r
465         print 'done'
466         Gratia.Send(r)
467
468
469
470
471 if __name__ == '__main__':
472     try:
473         Gratia.Initialize('/etc/gratia/awsvm/ProbeConfig')
474         GratiaWrapper.CheckPreconditions()
475         vmProbe=Awsgratiaprobe()

```

```

476     Filelock="filelock"
477     conf=GratiaCore.Config
478     duplicatelock=conf.getConfigAttribute("ExemptDuplicates")
479     filelock=conf.getConfigAttribute("DuplicateFilelock")
480     if duplicatelock == "True":
481         if os.path.isfile(Filelock):
482             fl=open(Filelock, 'r+')
483             date=datetime.datetime.now()
484             line=fl.readline()
485             print line
486             prevdate = datetime.datetime.strptime(line, "%Y-%m-%d %H:%M:%S.%f")
487             print prevdate
488             currtime=time.mktime(date.timetuple())
489             prevtime=time.mktime(prevdate.timetuple())
490             diff=currtime-prevtime
491             print diff
492             if diff>=3599.0:
493                 fl.seek(0, 0)
494                 fl.truncate()
495                 fl.write(str(date));
496                 t = os.path.getmtime(Filelock)
497                 print t
498                 print datetime.datetime.fromtimestamp(t)
499                 fl.close()
500             vmProbe.process()
501         else:
502             print "hour is not over yet"
503             fl.close()
504     else:
505         fl=open(Filelock, 'w+')
506         date=str(datetime.datetime.now())
507         fl.write(date);
508         fl.close()
509         vmProbe.process()
510     else:
511         vmProbe.process()
512 except Exception, e:
513     print >> sys.stderr, str(e)

```

```

514         sys.exit(1)
515     sys.exit(0)
516
517
518 -----
519 -----
520 -----
521
522
523 File: monitoringaccountingbilling/trunk/src/gratia/awsvm/awsvm_probe.cron.sh
524 python /usr/share/gratia/awsvm/aws-gratia-probe
525
526 -----
527 -----
528 -----
529
530
531 File: monitoringaccountingbilling/trunk/src/gratia/awsvm/hardwareinst
532 instance-type  vcpu    ram price
533 m3.medium      1   3.75  0.067
534 m3.large       2   7.5   0.133
535 t2.micro        1   1     0.013
536 m3.xlarge      4   15    0.266
537 t2.small       1   2     0.026
538 t2.medium      2   4     0.052
539 t2.large       2   8     0.104
540 m4.large       2   8     0.126
541 m4.2xlarge     8   32    0.504
542 m4.xlarge      4   16    0.252
543 m4.4xlarge     16  64    1.008
544 m4.10xlarge   40  160   2.52
545 m3.2xlarge     8   30    0.532
546 c4.large       2   3.75  0.11
547 c4.xlarge      4   7.5   0.22
548 c4.2xlarge     8   15    0.441
549 c4.4xlarge     16  30    0.882
550 c4.8xlarge     36  60    1.763
551 c3.large       2   3.75  0.105

```

```

552 c3.xlarge      4   7.5   0.21
553 c3.2xlarge     8   15    0.42
554 c3.4xlarge     16  30    0.84
555 c3.8xlarge     32  60    1.68
556 g2.2xlarge     8   15    0.65
557 g2.8xlarge     32  60    2.6
558 r3.large       2   15    0.175
559 r3.xlarge      4   30.5  0.35
560 r3.2xlarge     8   61    0.7
561 r3.4xlarge     16  122   1.4
562 r3.8xlarge     32  244   2.8
563 i1.xlarge      4   30.5  0.853
564 i2.2xlarge     8   61    1.705
565 i2.4xlarge     16  122   3.41
566 i2.8xlarge     32  244   6.82
567 d2.xlarge      4   30.5  0.69
568 d2.2xlarge     8   61    1.38
569 d2.4xlarge     16  56    2.76
570 d2.8xlarge     36  244   5.52
571
572 -----
573 -----
574 -----
575
576
577 File: monitoringaccountingbilling/trunk/src/gratia/awsvm/README
578 Steps to install AWS-gratia-probe(procedures for SL6)(not supported for SL5):
579 You could also follow
580 ... https://twiki.grid.iu.edu/bin/view/Documentation/Release3/InstallAwsVmGratiaProbe
581
582 1. First install the EPEL 6
583     rpm -Uvh http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
584
585 2. Next you need to get the osg repo
586     http://repo.grid.iu.edu/osg/3.2/osg-3.2-el6-release-latest.rpm
587     and use yum to install the gratia-probe-awsvm.
588     The rpm name being gratia-probe-awsvm-(% VERSION).el6.noarch.rpm
589
590 3. You have to install pip and boto3
591 B) yum install -y python-pip

```

```

589 C) Next install the boto3 package using pip
590     pip install boto3
591 4. Boto3 needs the aws credentials file containing the IAM Access ID and Key in order to access the
... AWS console.
592 A) create a credentials file in the standard place specified by AWS
593 5. Next modify the ProbeConfig file
594     vi /etc/gratia/awsvm/ProbeConfig
595 A) Modify the host name and ports to point to the gratia server like below
596     CollectorHost="fermicloud054.fnal.gov:8880"
597         SSLHost="fermicloud054.fnal.gov:8443"
598         SSLRegistrationHost="fermicloud054.fnal.gov:8080"
599 B) Also enable the probe by setting below to one
600     EnableProbe="1"
601 c) Also enable Exempt duplicates in order to prevent probe from sending records more than once
... per hour using
602     ExemptDuplicates="1"
603     Save the changes to config file
604 6. Next we need to add the probe's cron to the system cron and enable it
605 A) add the cron
606     chkconfig --add gratia-probes-cron
607 B) Enable the cron
608     chkconfig gratia-probes-cron on
609 c) Check its status
610     chkconfig --list gratia-probes-cron
611 D) also start the service
612     service gratia-probes-cron start
613 7. The aws-gratia-probe is started and will run every one hour
614     we can check the logs in
615     vi /var/logs/gratia/(%date).log
616
617
618
619 ALSO INCASE ANY ERRORS ARE THROWN BY THE PROBE DURING RUNTIME
620 OR IF THE PROBE DOESNT EXECUTE PROPERLY
621 1. IF the exempt duplicates is set and the filelock throws errors
622     check whether the filelock file exists at
623     /var/lib/gratia/filelock
624     and also where it is set properly in ProbeConfig file

```

```

625     DuplicateFilelock="/var/lib/gratia/filelock"
626 2. If it says hardwareinst doesnt exist check whether the file exists and the value is it properly
... in probe
627     HardwareDetails="/usr/share/gratia/awsvm/hardwareinst"
628 3. If u need to add or update the values to the table in hardwareinst plf follow the syntax
629     instance-type(\tab)vcpu(\tab)ram(\tab)price(\tab)(\newline)
630     value1(\tab)value2(\tab)value3(\tab)value4(\tab)(\newline)
631     eg : m3.medium(\tab)1(\tab)3.75(\tab)0.067(\tab)(\newline)
632
633
634 -----
635 -----
636 -----
637
638
639 File: monitoringaccountingbilling/trunk/src/gratia/site-packages/gratia/awsvm/__init__.py
640
641 -----
642 -----
643 -----
644
645
646 File: monitoringaccountingbilling/trunk/src/gratia/site-packages/gratia/awsvm/cpuutil.py
647 #!/usr/bin/env python
648 from pprint import pprint
649 import datetime;
650 import boto3;
651 import sys
652 import os
653 import time
654
655 class cpuUtilization:
656     def __init__(self):
657         self.ec2=boto3.client('ec2', region_name='us-west-2')
658     def getUtilPercent(self, instid):
659         ec2=boto3.client('ec2', region_name='us-west-2')
660         cw = boto3.client('cloudwatch', region_name='us-west-2')
661         resp=ec2.describe_instances(InstanceIds=[instid])

```

```

662     #pprint(resp)
663     #print 'hello'
664     resv=resp['Reservations']
665     for reservation in resv:
666         #pprint(reservation)
667         instances=reservation['Instances']
668         for instance in instances:
669             #pprint(instance)
670             #print instance['LaunchTime']
671             launchtime=instance['LaunchTime']
672             zone=instance['Placement']['AvailabilityZone']
673             #print zone
674             inst_type=instance['InstanceType']
675             print inst_type
676         #print launchtime.hour
677         minu=launchtime.minute
678         #print minu
679         EndTime=datetime.datetime.utcnow()
680         EndTime=EndTime.replace(minute=minu)
681         StartTime=EndTime.replace(hour=(EndTime.hour-1))
682         print StartTime
683         print EndTime
684
685     response = cw.get_metric_statistics(
686         Namespace='AWS/EC2',
687         MetricName='CPUUtilization',
688         Dimensions=[
689             {
690                 'Name': 'InstanceId',
691                 'Value': instid
692             },
693         ],
694         StartTime=StartTime,
695         EndTime=EndTime,
696         Period=3600,
697         Statistics=['Average', 'Minimum', 'Maximum'],
698         Unit='Percent')
699     #pprint(response)

```

```

700     datapoints=response['Datapoints']
701     if len(datapoints)==1:
702         datapoint=datapoints[0]
703         average=datapoint['Average']
704         #print average
705         return average
706
707
708 -----
709 -----
710 -----
711
712
713 File: monitoringaccountingbilling/trunk/src/gratia/site-packages/gratia/awsvm/get_account_id.py
714 #!/usr/bin/env python
715 from pprint import pprint;
716 import datetime;
717 import boto3;
718 import sys
719 import os
720 import time
721
722 class get_account_id:
723     def __init__(self):
724         self.ec2=boto3.client('ec2',region_name='us-west-2')
725
726     def get_id(self):
727         myec2=boto3.client('ec2',region_name='us-west-2')
728         resp=myec2.describe_images(Owners=['self'])
729         #pprint(resp)
730         images=resp['Images']
731         for image in images:
732             myowner=image['OwnerId']
733             #as written this will return the last OwnerID
734             #but they should all be the same
735         return myowner
736
737 -----

```

```

738 -----
739 -----
740
741
742 File: monitoringaccountingbilling/trunk/src/gratia/site-packages/gratia/awsvm/inst_hardware.py
743 #!/usr/bin/env python
744 from pprint import pprint
745 import datetime;
746
747 class insthardware:
748     def __init__(self,filelocation):
749         self.fileloc=filelocation
750     def gettypedetails(self):
751         types=[]
752         field=[]
753         #repo = {}
754
755         infile = open(self.fileloc,'r')
756         firstline = infile.readline()
757         fields=firstline.split("\t")
758         #print fields
759         for f in fields:
760             #print f
761             if f == "\n":
762                 fields.remove(f)
763
764             #print fields
765         lines= infile.readlines()
766         #print firstline
767         for i in lines:
768             #print i
769             values=i.split("\t")
770             values.remove("\n")
771             #print values
772             x=0
773             repo={}
774             while x<len(fields):
775                 repo[fields[x]]=values[x]

```

```

776             x+=1
777             #print repo
778             types.append(repo)
779             #pprint(types)
780             return types
781
782             #module = ''.join(i.split(',')[:-1])
783             #time = ''.join(i.split(',')[:1]).replace('\n','')
784             #if not module in repo:
785             #    repo[module] = time
786
787 -----
788 -----
789 -----
790
791
792 File: monitoringaccountingbilling/trunk/src/gratia/site-packages/gratia/awsvm/spot_price.py
793 #!/usr/bin/env python
794 import datetime;
795 import boto3;
796 import sys
797 import os
798 import time
799
800 class spot_price:
801     def __init__(self):
802         self.ec2=boto3.client('ec2',region_name='us-west-2')
803
804     def get_price(self,instid):
805         resp=self.ec2.describe_instances(InstanceIds=[instid])
806         #pprint(resp)
807         #print 'hello'
808         resv=resp['Reservations']
809         for reservation in resv:
810             #pprint(reservation)
811             instances=reservation['Instances']
812             for instance in instances:
813                 #pprint(instance)

```

```

814         #print instance['LaunchTime']
815         launchtime=instance['LaunchTime']
816         zone=instance['Placement']['AvailabilityZone']
817         #print zone
818         inst_type=instance['InstanceType']
819         #print inst_type
820     #print launchtime.hour
821     minu=launchtime.minute
822     #print minu
823     StartTime=datetime.datetime.utcnow()
824     #print StartTime
825     if(minu>StartTime.minute):
826         StartTime=StartTime.replace(hour=(StartTime.hour-1))
827     StartTime=StartTime.replace(minute=minu)
828     #print StartTime
829     EndTime=StartTime
830     #response =
... self.ec2.describe_spot_price_history(StartTime=datetime(2015,7,8,9,00,00),EndTime=datetime(2015,7,8
... ,9,00,00),InstanceTypes=['m3.medium'],ProductDescription=['Linux/UNIX'],AvailabilityZone='us-west-
... 2a',NextToken='abc')
831     #pprint(response)
832     #print 'hello'
833     resp1 = self.ec2.describe_spot_price_history(
834         DryRun=False,
835         StartTime=StartTime,
836         EndTime=EndTime,
837         InstanceTypes=[inst_type],
838         ProductDescriptions=['Linux/UNIX'],
839         Filters=[],
840         AvailabilityZone=zone,
841         MaxResults=1000,
842         NextToken='')
843     #pprint(resp1)
844     sphs=resp1['SpotPriceHistory']
845     if len(sphs)==0:
846         print "no spot price history as Instance is not of a spot Instance type"
847         spotprice=0
848     else:

```

```

849         sph=sphs[0]
850         spotprice=sph['SpotPrice']
851     #print spotprice
852     #print 'hello'
853     return spotprice
854     return StartTime
855 if __name__ == '__main__':
856     try:
857         sp=spot_price()
858         value=sp.get_price('i-d0952f26')
859         #print value
860     except IndexError:
861         print "The instance is not a spot type instance and Hence there is no spot price history"
862     except Exception, e:
863         print >> sys.stderr, str(e)
864         sys.exit(1)
865     sys.exit(0)
866
867 -----
868 -----
869 -----
870
871
872 File: monitoringaccountingbilling/trunk/src/monitor/aws.py
873 #!/usr/bin/python
874 from collections import defaultdict
875 from optparse import OptionParser
876 import time
877 import logging
878 import datetime
879
880 import boto3
881
882 from graphite import Graphite
883
884 def get_ec2_instance_cpu(region,instance,end_time=None,period=300):
885     if end_time is None:
886         end_time=datetime.datetime.utcnow()

```

```

887     cw = boto3.client('cloudwatch',region_name=region)
888     response = cw.get_metric_statistics(
889         Namespace='AWS/EC2',
890         MetricName='CPUUtilization',
891         Dimensions=[{'Name':"InstanceId",'Value':instance}],
892         StartTime=end_time-datetime.timedelta(seconds=period*2),
893         EndTime=end_time,
894         Period=period,
895         Statistics=['Average','Minimum','Maximum'],
896         Unit='Percent')
897     datapoints=response['Datapoints']
898     r = {}
899     if len(datapoints) > 0:
900         datapoint=datapoints[-1]
901         r['avg'] = datapoint['Average']
902         r['min'] = datapoint['Minimum']
903         r['max'] = datapoint['Maximum']
904     else:
905         logging.warning('no CPU utilization received for instance %s'%instance)
906     return r
907
908 def get_ec2_instances(region):
909     r = defaultdict(int)
910     cpu = defaultdict(float)
911     #try:
912     ec2 = boto3.resource('ec2',region)
913     instances = ec2.instances.all()
914     for i in instances:
915         type = i.instance_type.replace('.', '_')
916         if i.state['Name'] == 'running':
917             cpu_usage = get_ec2_instance_cpu(region, i.instance_id)
918             cpu[type] += cpu_usage.get('avg',0)
919             metric = "{0}.counts.{1}".format(type,i.state['Name'])
920             r[metric] += 1
921
922     for k,v in cpu.iteritems():
923         r[k+".cpu.avg"] = v/r[k+".counts.running"]
924     #except Exception as e:

```

```

925     # logging.error('error communicating with AWS: %s'%e)
926     # raise e
927     return r
928
929 if __name__ == '__main__':
930     parser = OptionParser()
931     parser.add_option('-t','--test',action="store_true",
932         dest="test",default=False,
933         help="output data to stdout, don't send to graphite. Implies -1.")
934     parser.add_option('-1','--once',action="store_true",
935         dest="once",default=False,
936         help="run once and exit")
937     parser.add_option('-r','--region', default="us-west-2",
938         help="AWS region to query; default=us-west-2")
939     parser.add_option('-n','--namespace', default="test",
940         help="base graphite metric namespace; default=test")
941     parser.add_option('-m','--meta_namespace', default="probes.aws_instances",
942         help="probe metadata graphite metric namespace; default=probes.aws_instances")
943     parser.add_option('-i','--interval', type="int", default=240,
944         help="post interval (seconds); default=240")
945
946     (opts,args) = parser.parse_args()
947
948     logformat="%asctime)s - %(name)s - %(levelname)s - %(message)s"
949     if opts.test:
950         logging.basicConfig(format=logformat,level=logging.DEBUG)
951     else:
952         logging.basicConfig(format=logformat,level=logging.INFO)
953
954     g = Graphite()
955     while True:
956         start = time.time()
957         data = get_ec2_instances(opts.region)
958         duration = time.time()-start
959         logging.info("queried AWS region {0} in {1} s".format(opts.region,duration))
960
961         send_start = time.time()
962         g.send_dict(opts.namespace, data, debug_print=opts.test, send_data=(not opts.test))

```

```
963     meta_data = {
964         "update_time": duration,
965     }
966     g.send_dict(opts.meta_namespace, meta_data, debug_print=opts.test, send_data=(not
... opts.test))
967
968     duration = time.time()-send_start
969     logging.info("sent {0} metrics to graphite namespace {1} in {2}
... s".format(len(data)+len(meta_data), opts.namespace, duration))
970
971     if opts.test or opts.once:
972         break
973
974     duration = time.time()-start
975     sleep = max(opts.interval-duration-10,0)
976     logging.info("sleeping {0} s".format(sleep))
977     time.sleep(sleep)
978
979 -----
980 -----
981 -----
982
983
984
```

```

1 File: ondemandservices/hepcloud-init-workernode
2 #!/bin/sh
3 ### BEGIN INIT INFO
4 # chkconfig: 2345 27 25
5 # Provides:      hepcloud-init-workernode
6 # Required-Start: $local_fs $network
7 # Should-Start:  $time
8 # Required-Stop:
9 # Should-Stop:
10 # Default-Start:  2 3 4 5
11 # Default-Stop:   0 1 6
12 # Short-Description: Fix host name and az-specific config files
13 # Description:     Start cloud-init and runs the initialization phase
14 #                 and any associated initial modules as desired.
15 #test
16 ### END INIT INFO
17 LOG="/var/log/hepcloud-init-workernode.log"
18 RETVAL=0
19
20 prog="hepcloud-init-workernode"
21
22 start() {
23
24 touch /var/lock/subsys/hepcloud-init-workernode
25 #
26 # get the public hostname of the EC2 instance and change the
27 # output of the hostname command to match that. (needed for gridftp).
28 #
29 mypublicip=`GET http://169.254.169.254/latest/meta-data/public-ipv4`
30 myrc=$?
31 if [ $myrc -ne 0 ]
32 then
33     echo "My public IP not defined" >> $LOG
34     return 6
35 fi
36 nslookup $mypublicip | grep "name =" | awk -F ' ' '{print $4}' | sed 's/com\./com/' > /etc/hostname
37 myrc=$?
38 if [ $myrc -ne 0 ]

```

```

39 then
40     echo "My public DNS name not defined" >> $LOG
41     return 7
42 fi
43 hostname -F /etc/hostname
44
45 # This script modifies the CVMFS and Frontier scripts on VM startup
46 # to point to the ELB-enabled squid stack for the respective
47 # availability zone
48
49 zone=$(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone)
50 echo $zone
51 echo $zone >> $LOG
52 addr="http://elb2.$zone.elb.fnaldata.org:3128" #HK> export http_proxy does not like two
... back-slashes, I had to remove them
53
54 # new code by ST and HK #####
55 export http_proxy=$addr
56 wget http://cvmfs.fnal.gov:8000/cvmfs/cms.cern.ch/.cvmfspublished
57 returnvalue=$?
58 if [ $returnvalue -ne 0 ]
59 then
60     echo "Squid Server is not accessible in $zone, trying us-west-2b" >> $LOG
61
62     if [ $zone != "us-west-2b" ]
63     then
64         uswest2baddr="http://elb2.us-west-2b.elb.fnaldata.org:3128"
65         export http_proxy=$uswest2baddr
66         wget http://cvmfs.fnal.gov:8000/cvmfs/cms.cern.ch/.cvmfspublished
67         returnvalue=$?
68         if [ $returnvalue -ne 0 ]
69         then
70             echo "Squid Server is not accessible at all" >> $LOG
71             return 11
72         else
73             echo "Squid Server is available in us-west-2b" >> $LOG
74             addr="\elb2.us-west-2b.elb.fnaldata.org:3128"
75         fi

```

```

76
77     else
78         echo "Squid Server is not available even in us-west-2b" >> $LOG
79         return 11
80     fi
81
82 else
83     echo "Squid Server is available in $zone" >> $LOG
84     addr="http://elb2.$zone.elb.fnaldata.org:3128" #HK> now, sed command below requires two
... back-slashes, I had restore them here.
85 fi
86 # END: new code by HK and ST #####
87
88
89 # make sure /etc/cvmfs/default.local is in place
90 if [ ! -r /etc/cvmfs/default.local ]
91 then
92     echo "/etc/cvmfs/default.local not found" >> $LOG
93     return 8
94 fi
95 # and modify
96 sed -i -e "s/(CVMFS_HTTP_PROXY=\\).*/\\1$addr/" /etc/cvmfs/default.local
97
98
99
100 # make sure /usr/bin/cvmfs_config is in place
101 if [ ! -x /usr/bin/cvmfs_config ]
102 then
103     echo "/usr/bin/cvmfs_config not executable" >> $LOG
104     return 9
105 fi
106 # and run it
107 /usr/bin/cvmfs_config reload >> $LOG 2>&1
108
109
110
111 # make sure /etc/cvmfs/SITECONF/local/JobConfig/site-local-config.xml is in place
112 if [ ! -r /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml ]

```

```

113 then
114     echo "/etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml not found" >> $LOG
115     return 10
116 fi
117 # and modify
118 sed -i -e "s/(<proxy url=\\).*/\\1"$addr"/>/"
... /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml
119
120
121
122 # getting rid of the Fermi-specific crlsquid.fnal.gov and crl-cache.fnal.gov if it's there
123 if [ -r /etc/fermi-crl.conf ]
124 then
125     cp -p /etc/fermi-crl.conf /etc/fermi-crl.conf.fermisav
126     sed -i -e "s/(http_proxy = \\).*/\\1$addr/" /etc/fermi-crl.conf
127     grep -v prepend_url /etc/fermi-crl.conf > /etc/fermi-crl.conf.temp
128     cp /etc/fermi-crl.conf.temp /etc/fermi-crl.conf
129 # start a fetch-crl now at S27 instead of enabling the fetch-crl-boot
130 # and throw it into background.
131
132 # commented out by HK/ST in order to make sure fetch-crl does not populate
... /etc/grid-security/certificates/ with *.r0 files
133 # nohup /usr/sbin/fermi-crl < /dev/null >> $LOG 2>&1 &
134 fi
135
136 return 0
137 }
138
139 stop() {
140
141 rm /var/lock/subsys/hepcloud-init-workernode
142
143 return 0
144 }
145
146 case "$1" in
147     start)
148         start

```

```

149     RETVAL=$?
150 ;;
151 stop)
152     stop
153     RETVAL=$?
154 ;;
155 restart|try-restart|condrestart)
156     ## Stop the service and regardless of whether it was
157     ## running or not, start it again.
158     #
159     ## Note: try-restart is now part of LSB (as of 1.9).
160     ## RH has a similar command named condrestart.
161     start
162     RETVAL=$?
163 ;;
164 reload|force-reload)
165     # It does not support reload
166     RETVAL=3
167 ;;
168 status)
169     echo -n $"Checking for service $prog:"
170     # Return value is slightly different for the status command:
171     # 0 - service up and running
172     # 1 - service dead, but /var/run/ pid file exists
173     # 2 - service dead, but /var/lock/ lock file exists
174     # 3 - service not running (unused)
175     # 4 - service status unknown :-(
176     # 5--199 reserved (5--99 LSB, 100--149 distro, 150--199 appl.)
177     RETVAL=3
178 ;;
179 *)
180     echo "Usage: $0 {start|stop|status|try-restart|condrestart|restart|force-reload|reload}"
181     RETVAL=3
182 ;;
183 esac
184
185 exit $RETVAL
186

```

```

187 -----
188 -----
189 -----
190
191
192 File: ondemandservices/largequery.sh
193 #!/bin/bash
194
195 #This script should be present in the /root directory to work.If you are planning to place to
... somewhere else please change the path in the lines 6-8
196
197 zone=$(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone)
198
199 echo zone
200
201 addr="http://elb2.$zone.elb.fnaldata.org:3128"
202
203 sed -i -e "s/(CVMFS_HTTP_PROXY=).*\/1$addr/" ../etc/cvmfs/default.local
204
205 export http_proxy=http://elb2.us-west-2b.elb.fnaldata.org:3128
206
207 export STRATUM1URL="http://cvmfs.fnal.gov:8000"
208
209 NUMPROCESSES=25
210
211 WGETSPERITER=100
212
213 let I=0
214
215 #RUNNING="running.`uname -n`"
216
217 #trap "rm -f $RUNNING" 0
218
219 #touch $RUNNING
220
221 let P=0
222
223 while [ $P -lt $NUMPROCESSES ]; do

```

```

224
225     let P=$P+1
226
227         ( let I=$I+1
228
229             echo "Iteration $P:$I"
230
231             time bash -c 'n=0; while [ $n -lt "$WGGETSPERITER" ]; do wget -qO/dev/null
... "$STRATUM1URL/cvmfs/fermilab.opensciencegrid.org/data/3d/d8c5d91b8a94cd26a734c48188d5bbc223855bP";
... let n=$n+1; done' ) > /tmp/$P.out 2>&1 &
232
233 done
234
235 wait
236
237
238 -----
239 -----
240 -----
241
242
243 File: ondemandservices/smallquery.sh
244 #!/bin/bash
245
246 #This script should be present in the /root directory to work.If you are planning to place to
... somewhere else please change the path in the lines 6-8
247
248 zone=$(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone)
249
250 echo zone
251
252 addr="http://\$/elb2.$zone.elb.fnaldata.org:3128"
253
254 export http_proxy=$addr
255
256 export STRATUM1URL="http://cvmfs.fnal.gov:80"
257
258 NUMPROCESSES=25

```

```

259
260 WGETSPERITER=25
261
262 REPEATSPERWGET=500
263
264 let I=0
265
266 #RUNNING="running.`uname -n`"
267
268 #trap "rm -f $RUNNING" 0
269
270 #touch $RUNNING
271
272 let P=0
273
274 URL="$STRATUM1URL/cvmfs/fermilab.opensciencegrid.org/.cvmfspublished"
275
276 let U=0
277
278 export URLs=""
279
280 while [ $U -lt $REPEATSPERWGET ]; do
281
282     let U=$U+1
283
284     URLs="$URLS $URL"
285
286 done
287
288 while [ $P -lt $NUMPROCESSES ]; do
289
290     let P=$P+1
291
292     # while [ -f $RUNNING ]; do
293
294         ( let I=$I+1
295
296             echo "Iteration $P:$I"

```

```
297
298         time bash -c 'n=0; while [ $n -lt "$WGGETSPERITER" ];
299
300 do wget -qO/dev/null $URLS; let n=$n+1; done' ) > /tmp/$P.out 2>&1 &
301
302     # exit
303
304 #         done &
305
306 done
307
308 wait
309
310 iftop
311
312
313
314
315 -----
316 -----
317 -----
318
319
320 File: ondemandservices/squid-client-addr-change.sh
321 #!/bin/bash
322 #This script should be present in the /root directory to work.If you are planning to place to
...
somewhere else please change the path in the lines 6-8
323 zone=$(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone)
324 echo zone
325 addr="http://elb2.$zone.elb.fna1data.org:3128"
326 sed -i -e "s/(CVMFS_HTTP_PROXY=\\).*\/\1$addr/" ../etc/cvmfs/default.local
327 sed -i -e "s/(http_proxy=\\).*\/\1$addr/" ../cvmfsload/smallquery
328 sed -i -e "s/(http_proxy=\\).*\/\1$addr/" ../cvmfsload/largequery
329
330
331 -----
332 -----
333 -----
```

```

1 | File: spotpricehistory/SpotPriceHistory.py
2 |
3 | import boto3
4 | import datetime
5 | import os.path
6 | from boto3.session import Session
7 |
8 | class SpotPriceHistory:
9 |     '''
10 |    This class is used for getting spot pricing history
11 |    '''
12 |
13 |    startTime = ""
14 |    endTime = ""
15 |    zone = "us-west-2a"
16 |    instanceType="m3.medium"
17 |    os="Linux/UNIX"
18 |    #historyData = {}
19 |    #dataList=[]
20 |    nextToken=""
21 |    def __init__(self,instanceType,zone):
22 |        self.instanceType=instanceType
23 |        self.zone=zone
24 |        self.historyData={}
25 |        self.dataList=[]
26 |        self.filename="Database/"+self.instanceType+"_"+self.zone
27 |        self.readLastTimeFromDatabase()
28 |
29 |    def set_startTime(self,startTime):
30 |        self.startTime=startTime
31 |    def set_endTime(self,endTime):
32 |        self.endTime=endTime
33 |    def set_zone(self,zone):
34 |        self.zone=zone
35 |    def set_instanceType(self,instanceType):
36 |        self.instanceType=instanceType
37 |    def set_os(self,os):
38 |        self.os=os
39 |
40 |
41 |    def getSpotPriceHistory(self):
42 |        session = Session(aws_access_key_id='',
43 |                          aws_secret_access_key='',
44 |                          region_name=self.zone[:-1])
45 |        client = session.client('ec2')
46 |        iterates=0
47 |        while iterates==0 or self.nextToken!="":
48 |            temp = client.describe_spot_price_history(
49 |                DryRun=False,
50 |                StartTime=self.startTime,
51 |                EndTime=self.endTime,
52 |                InstanceTypes=[self.instanceType],
53 |                ProductDescriptions=[self.os],
54 |                Filters=[],
55 |                AvailabilityZone= self.zone,
56 |                MaxResults=1000,
57 |                NextToken=self.nextToken
58 |            )
59 |            self.dataList.insert(0, temp)
60 |            # tempDic=self.historyData.copy()
61 |            # tempDic.update(temp)
62 |            # self.historyData=tempDic
63 |            self.nextToken=temp['NextToken']
64 |            iterates+=1
65 |    def printHistoryData(self):
66 |        for dicts in self.dataList:
67 |            for i in reversed(dicts['SpotPriceHistory']):
68 |                print
69 |                ... (i['InstanceType'],i['ProductDescription'],i['SpotPrice'],str(i['Timestamp']),i['
70 |                ... AvailabilityZone'])
71 |
72 |    def getCredentials(self):
73 |        '''
74 |        Get AWS credentials from file
75 |        '''

```

```

75     def writeHistoryData(self):
76         '''
77         Write the historical data into database
78         '''
79         filename=self.filename
80         if not os.path.isfile(filename):
81             f=open(filename,"w")
82             f.write("DateTime Price InstanceType Zone\n")
83             for dicts in self.dataList :
84                 for i in reversed(dicts['SpotPriceHistory']):
85                     f.write(i['Timestamp'].strftime("%Y-%m-%dT%H:%M:%S.%f")+
... "+str(i['SpotPrice'])+" "+str(i['InstanceType'])+" "+str(i['AvailabilityZone'])+"\n")
86                     f.close()
87             else:
88                 f=open(filename,"a")
89                 for dicts in self.dataList :
90                     for i in reversed(dicts['SpotPriceHistory']):
91 #                         for t in i['Timestamp'].timetuple():
92 #                             print t
93 #                         for tt in self.startTime.timetuple():
94 #                             print tt
95                             if i['Timestamp'].replace(tzinfo=None)>self.startTime:
96                                 f.write(i['Timestamp'].strftime("%Y-%m-%dT%H:%M:%S.%f")+
... "+str(i['SpotPrice'])+" "+str(i['InstanceType'])+" "+str(i['AvailabilityZone'])+"\n")
97                                 f.close()
98
99
100     def readLastTimeFromDatabase(self):
101         '''
102         read last time stamp from the Database, and set start time and end time
103         '''
104         self.endTime=datetime.datetime.utcnow()
105         filename = self.filename
106         if not os.path.isfile(filename):
107             print ("File does not exist! Start from 90 days ago!")
108             self.startTime=self.endTime-datetime.timedelta(days=90)
109
110         else:

```

```

111         with open(filename,"r") as f:
112             for lines in f:
113                 pass
114                 last=lines
115                 #content=f.read().splitlines()
116                 #tempStr=content[len(content)-2].split(" ")
117                 tempStr=last.split(" ")
118                 print "Last time stamp: " + tempStr[0]
119                 self.startTime=datetime.datetime.strptime(tempStr[0],"%Y-%m-%dT%H:%M:%S.%f")
120                 f.close()
121 #             self.startTime=self.startTime.replace(tzinfo=None)
122 #             for tt in self.startTime.timetuple():
123 #                 print tt
124
125
126 vCPUs = {
127     'c3.large':2,
128     'c3.xlarge':4,
129     'c3.2xlarge':8,
130     'c3.4xlarge':16,
131     'c3.8xlarge':32,
132     'm3.medium':1,
133     'm3.large':2,
134     'm3.xlarge':4,
135     'm3.2xlarge':8,
136 }
137
138 ecu = {
139     'c3.large':7,
140     'c3.xlarge':14,
141     'c3.2xlarge':28,
142     'c3.4xlarge':55,
143     'c3.8xlarge':108,
144     'm3.medium':3,
145     'm3.large':6.5,
146     'm3.xlarge':13,
147     'm3.2xlarge':26,
148 }

```

```

149
150 std_prices = {
151     'c3.large' :0.105,
152     'c3.xlarge' :0.210,
153     'c3.2xlarge':0.420,
154     'c3.4xlarge':0.840,
155     'c3.8xlarge':1.680,
156     'm3.medium' :0.070,
157     'm3.large' :0.140,
158     'm3.xlarge' :0.280,
159     'm3.2xlarge':0.560
160 }
161
162 -----
163 -----
164 -----
165
166
167 File: spotpricehistory/updateDatabase.py
168 from SpotPriceHistory import SpotPriceHistory
169 import datetime
170 import os.path
171 #from analysis import simulation
172
173 instances=["m3.2xlarge","c3.2xlarge","m3.xlarge","c3.xlarge","m3.medium","m3.large","m3.xlarge"
...
,"c3.large","c3.4xlarge","c3.8xlarge"]
174 zone=["us-east-1b",
...
"us-east-1c","us-east-1d","us-east-1e","us-west-1a","us-west-1c","us-west-2a",
...
"us-west-2b","us-west-2c",]
175
176 if not os.path.exists("Database"):
177     os.mkdir("Database")
178 if not os.path.exists("Histogram"):
179     os.mkdir("Histogram")
180
181
182 for i in instances:
183     for z in zone:

```

```

184         awsPrice=SpotPriceHistory(i,z)
185         awsPrice.getSpotPriceHistory()
186         awsPrice.writeHistoryData()
187     #     analyze=simulation(i,z)
188     #     analyze.writeHistogram()
189     print i + " in " + z + " finishes!"
190
191 -----
192 -----
193 -----
194
195
196

```

```

1 | File: imagemanagement/step2/config.py
2 | import os
3 |
4 | import ini_handler
5 |
6 | from errors import ConfigError
7 |
8 | from simple_logging import Logger
9 | from simple_logging import FileWriter
10 | from simple_logging import SysLogWriter
11 | from simple_logging import ConsoleWriter
12 |
13 | from contextualization_types import CONTEXT_TYPE_EC2
14 | from contextualization_types import CONTEXT_TYPE_NIMBUS
15 | from contextualization_types import CONTEXT_TYPE_OPENNEBULA
16 |
17 | class Config(object):
18 |     valid_context_types = [CONTEXT_TYPE_EC2, CONTEXT_TYPE_NIMBUS, CONTEXT_TYPE_OPENNEBULA]
19 |
20 |     def __init__(self, config_ini="/etc/glideinwms/glidein-pilot.ini"):
21 |         if not os.path.exists(config_ini):
22 |             raise ConfigError("%s does not exist" % config_ini)
23 |
24 |         self.ini = ini_handler.Ini(config_ini)
25 |
26 |         self.default_max_lifetime = self.ini.get("DEFAULT", "default_max_lifetime", "172800")
... # 48 hours
27 |         self.max_lifetime = self.default_max_lifetime # can be overridden
28 |         self.disable_shutdown = self.ini.getBoolean("DEFAULT", "disable_shutdown", False)
29 |         self.max_script_runtime = self.ini.get("DEFAULT", "max_script_runtime", "60")
30 |
31 |         self.pre_script_dir = self.ini.get("DIRECTORIES", "pre_script_dir",
... "/usr/libexec/glideinwms_pilot/PRE")
32 |         self.post_script_dir = self.ini.get("DIRECTORIES", "post_script_dir",
... "/usr/libexec/glideinwms_pilot/POST")
33 |
34 |         # home directory is created by the rpm
35 |         self.home_dir = "/home/glidein_pilot"

```

```

36 |         self.glidein_user = "glidein_pilot"
37 | #HK
38 |         self.scratch_dir = "/home/scratchgws"
39 |
40 |         # glidein_startup.sh specific attributes
41 |         self.factory_url = ""
42 |         self.pilot_args = ""
43 |         self.proxy_file = ""
44 |         self.pilot_args = ""
45 |
46 |     def setup(self):
47 |         self.setup_logging()
48 |         self.setup_pilot_files()
49 |         self.setup_contextualization()
50 |
51 |     def setup_pilot_files(self):
52 |         self.ini_file = "%s/glidein_userdata" % self.home_dir
53 |         self.userdata_file = "%s/userdata_file" % self.home_dir
54 |         self.log.log_info("Default ini file: %s" % self.ini_file)
55 |         self.log.log_info("Default userdata file: %s" % self.userdata_file)
56 |
57 |     def setup_contextualization(self):
58 |         self.contextualization_type = self.ini.get("DEFAULT", "contextualize_protocol")
59 |         self.log.log_info("Contextualization Type identified as: %s" %
... self.contextualization_type)
60 |         if self.contextualization_type in Config.valid_context_types:
61 |             if self.contextualization_type == CONTEXT_TYPE_EC2:
62 |                 self.ec2_url = self.ini.get("DEFAULT", "ec2_url")
63 |             elif self.contextualization_type == CONTEXT_TYPE_NIMBUS:
64 |                 self.nimbus_url_file = self.ini.get("DEFAULT", "nimbus_url_file")
65 |             elif self.contextualization_type == CONTEXT_TYPE_OPENNEBULA:
66 |                 self.one_user_data_file = self.ini.get("DEFAULT", "one_user_data_file")
67 |             else:
68 |                 raise ConfigError("configured context type not valid")
69 |
70 |     def setup_logging(self):
71 |         log_writer = None
72 |         log_writer_class = self.ini.get("DEFAULT", "logger_class", None)

```

```

73     if log_writer_class:
74         if log_writer_class == "SyslogWriter":
75             facility = self.ini.get("DEFAULT", "syslog_facility", None)
76             priority = self.ini.get("DEFAULT", "syslog_priority", None)
77             log_writer = SyslogWriter(facility=facility, priority=priority)
78         elif log_writer_class == "ConsoleWriter":
79             output = self.ini.get("DEFAULT", "console_output", "stdout")
80             log_writer = ConsoleWriter(output=output)
81         else:
82             log_writer = FileWriter(self.home_dir)
83     else:
84         #log_writer = FileWriter(self.home_dir)
85         log_writer = FileWriter('/var/log/glideinwms-pilot')
86     self.log = Logger(log_writer)
87     self.log.log_info("Pilot Launcher started...")
88
89 def export_custom_env(self):
90     """
91     @returns: string containing the shell (sh, bash, etc) directives to
92             export the environment variables
93     """
94     environment = ""
95     try:
96         env = self.get_custom_env()
97         for option in env:
98             environment += "export %s=%s; " % (option, env[option])
99     except:
100         # pylint: disable=W0702
101         pass
102     return environment
103
104 def get_custom_env(self):
105     """
106     Returns a dictionary of the parent process environment plus the custom
107     environment variables defined in the pilot config file.
108
109     NOTE: All custom environment variable names will be upper cased. The
110     values for the custom environment variables will not be modified.

```

```

111
112     @returns: dictionary containing the desired process environment
113     """
114     environment = {}
115     # inherit the parent process environment
116     for var in os.environ.keys():
117         environment[var] = os.environ[var]
118
119     try:
120         # add in the custom environment
121         for option in self.cp.ini.options("CUSTOM_ENVIRONMENT"):
122             environment[str(option).upper()] = self.ini.get("CUSTOM_ENVIRONMENT", option)
123     except:
124         # pylint: disable=W0702
125         pass
126
127     # Add in the pilot proxy
128     environment["X509_USER_PROXY"] = self.proxy_file
129     environment["HOME"] = self.home_dir
130     environment["LOGNAME"] = self.glidein_user
131
132     environment["SCRATCH"] = self.scratch_dir
133     return environment
134
135 -----
136 -----
137 -----
138
139
140 File: imagemanagement/step2/hkpilot.sh
141 #!/bin/bash
142 export http_proxy="http://131.225.148.121:3128"
143 export https_proxy="http://131.225.148.121:3128"
144
145 mkdir -p /etc/puppet/modules
146 #puppet module install desalvo-cvmfs
147
148 cat << EOF | puppet apply

```

```

149 class cvmfs {
150 }
151 class cvmfs::client (
152   \ $repositories = 'sft.cern.ch',
153   \ $quota_limit = 30000,
154   \ $http_proxy = undef,
155 ) inherits cvmfs {
156
157   package { cvmfs: ensure => installed, require => Package['osg-release'] }
158   package { cvmfs-config-osg: ensure => installed, require => Package['osg-release'] }
159   if (!defined(Package["fuse"])) {
160     package { fuse: ensure => latest }
161   }
162
163   if (!$http_proxy) {
164     \ $default_http_proxy = 'DIRECT'
165
166   } else {
167     \ $default_http_proxy = '${http_proxy};DIRECT'
168   }
169
170   file {
171     '/etc/cvmfs/default.local':
172     owner => root, group => root, mode => 644,
173     content => inline_template("CVMFS_REPOSITORIES=<%= repositories -%>
... \nCVMFS_QUOTA_LIMIT=<%= quota_limit -%> \nCVMFS_HTTP_PROXY=\"<%= default_http_proxy -%>\" \n
..."),
174     notify => Exec['cvmfs setup']
175   }
176
177   exec { 'cvmfs reload':
178     path => [ '/bin', '/usr/bin' ],
179     command => 'cvmfs_config reload',
180     timeout => 0,
181     refreshonly => true,
182     require => [Package['cvmfs'],Package['cvmfs-config-osg']]
183   }
184

```

```

185   exec { 'cvmfs setup':
186     path => [ '/bin', '/usr/bin' ],
187     command => 'cvmfs_config setup',
188     timeout => 0,
189     require => [Package['cvmfs'],Package['cvmfs-config-osg']]
190   }
191 }
192
193 group { 'cvmfs-gid':
194   name => 'cvmfs',
195   gid => '9125',
196   ensure => 'present',
197 }
198
199 user { 'cvmfs-uid':
200   name => 'cvmfs',
201   uid => '46084',
202   gid => '9125',
203   home => '/var/lib/cvmfs',
204   comment => 'CernVM File System',
205   shell => '/sbin/nologin',
206   ensure => 'present',
207   require => Group['cvmfs-gid'],
208 }
209
210 package { 'yum-conf-epel':
211   ensure => 'installed',
212 }
213
214 package { 'yum-plugin-priorities':
215   ensure => 'installed',
216 }
217
218 package { 'osg-release':
219   provider => 'rpm',
220   source => 'http://repo.grid.iu.edu/osg/3.2/osg-3.2-el6-release-latest.rpm',
221 }
222
223 yumrepo { 'epel':

```

```

223     enabled => 1,
224     require => Package['yum-conf-epel'],
225 }
226
227 package { 'osg-ca-certs':
228     ensure => 'installed',
229     require => Package['osg-release']
230 }
231
232 package { 'osg-oasis':
233     ensure => 'installed',
234     require => [ Package['osg-release'], User['cvmfs-uid'] ],
235 }
236
237 package { 'osg-wn-client':
238     ensure => 'installed',
239     require => Package['osg-release']
240 }
241
242 package { 'fetch-crl':
243     ensure => 'installed',
244     require => Package['yum-conf-epel']
245 }
246
247 service {'fetch-crl-boot':
248     ensure => 'running',
249     enable => 'false',
250     require => [ File['/etc/fetch-crl.conf'], Package['osg-wn-client'], ]
251 }
252
253 service {'fetch-crl-cron':
254     ensure => 'running',
255     enable => 'false',
256     require => [ File['/etc/fetch-crl.conf'], Package['osg-wn-client'], ]
257 }
258
259 file { '/etc/fetch-crl.conf': #Not sure this is not the default already....
260     ensure => file,

```

```

261     mode => '644',
262     owner => 'root',
263     group => 'root',
264     content => '#
265 ## PUPPET MAINTAINED file for fetch-crl
266 ##
267 infodir = /etc/grid-security/certificates
268 agingtolerance = 24
269 nosymlinks
270 nowarnings
271 noerrors
272 stateless
273 logmode = syslog
274 syslogfacility = daemon
275 http_proxy = http://crlsquid.fnal.gov:3128
276 prepend_url = http://crl-cache.fnal.gov/certificates/@ANCHORNAME.r@R@
277 ',
278     require => Package['fetch-crl']
279 }
280
281 class { 'cvmfs::client':
282     repositories =>
283     'oasis.opensciencegrid.org,atlas.cern.ch,atlas-condb.cern.ch,cms.cern.ch,lhcb.cern.ch,nova.
284     fnaldata.gov',
285     quota_limit => 4000,
286     http_proxy => 'squid.fnal.gov:3128',
287     require => Package['osg-oasis'],
288 }
289
290 file { '/etc/cvmfs/keys/fnaldata.gov.pub':
291     ensure => 'file',
292     content => "-----BEGIN PUBLIC
... KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAA0CAQ8AMIIBCgKCAQEAngbplxwU8YgAoTcF51cr\
... nmt28kSWUMbn8fj4kK0kwtHILc1Hsh6sWbrTL/iZhWJDFPyVqnIyDGEHDUkHSt9XE\
... nWgdoz6VXoHbiVNg3xLLE30RCycQumeb/wXf0juvL77LGvMmspmBB/2W/0Gprw7Kr\
... n4dWd9LHmz90Lx1WEJpGLsdzPzAvhqgAxlTYaIuB9pG/M+Rh5ep4ZRQArpvr16yxL\
... njBwLPavDrSbnJj0wqB27G0inBsKro5d8qeGYmNywC3cKP5aVZ9IoL/7XI7RLxEtp\
... nfrHmp1l9AmzD0qqAfGqPS1G5ziR0PB7evSJBBrP08An2o+w15I0CF89mjm9I8IsvW\n4QIDAQAB\n-----END PUBLIC

```

```

290... KEY-----\n",
291     require => Class['cvmfs::client'],
292     }
293
294     file { '/etc/cvmfs/domain.d/fnaldata.gov.conf':
295     ensure => 'file',
296     content =>
... "CVMFS_PUBLIC_KEY=/etc/cvmfs/keys/fnaldata.gov.pub\nCVMFS_SERVER_URL='http://cvmfss1data.fnal.
... gov:8000/cvmfs/@fqrn@;http://hcc-cvmfs.unl.edu:8000/cvmfs/@fqrn@\n\nCVMFS_HTTP_PROXY=DIRECT\
... nCVMFS_ALIEN_CACHE=/pnfs/fs/usr/cvmfs/fermicloud-alien-cache\nCVMFS_QUOTA_LIMIT=-1\
... nCVMFS_SHARED_CACHE=no\n",
297     require => Class['cvmfs::client'],
298     }
299
300     file { ['pnfs', '/pnfs/fs', '/pnfs/fs/usr', '/pnfs/fs/usr/cvmfs ']:
301     ensure => 'directory',
302     }
303
304     mount { '/pnfs/fs/usr/cvmfs':
305     device => 'pnfs-stken:/pnfs/fs/usr/cvmfs',
306     fstype => 'nfs4',
307     ensure => 'mounted',
308     options => 'minorversion=1',
309     atboot => 'true',
310     require => File['/pnfs/fs/usr/cvmfs'],
311     }
312
313     yumrepo {'glideinwms':
314     descr    => 'Glideinwms repository 6 - i\${basearch}',
315     baseurl  => 'http://fermigrid.fnal.gov/files/glideinwms/prod/6/\${basearch}/',
316     enabled  => 1,
317     gpgcheck => 0,;
318     'glideinwms-dev':
319     descr    => 'Glideinwms dev repository 6 - \${basearch}',
320     baseurl  => 'http://fermigrid.fnal.gov/files/glideinwms/dev/6/\${basearch}/',
321     enabled  => 0,
322     gpgcheck => 0,
323     }

```

```

324
325     package { 'glideinwms-vm-one':
326     ensure => 'installed',
327     require => Yumrepo['glideinwms'],
328     notify => Exec['/usr/sbin/ntpdate'],
329     }
330
331     service {'glideinwms-pilot':
332     ensure => 'running',
333     enable => 'true',
334     require => Exec['hktempoverwrite'],
335     }
336
337     exec { 'hktempoverwrite':
338     command => '/bin/mount -t iso9660 /dev/sr0 /mnt;/bin/cp -f /mnt/pilot-launcher
... /usr/sbin/pilot-launcher; /bin/cp -f /mnt/mount_ephemeral
... /usr/libexec/glideinwms_pilot/PRE/mount_ephemeral; /bin/cp -f /mnt/config.py
... /usr/lib/python2.6/site-packages/glideinwms_pilot/config.py',
339     require => Package['glideinwms-vm-one'],
340     }
341
342     exec {'/usr/sbin/ntpdate':
343     refreshonly => true,
344     command => '/usr/sbin/ntpdate -s 131.225.82.122',
345     require => [Service['glideinwms-pilot']]
346     }
347
348
349 EOF
350
351 -----
352 -----
353 -----
354 -----
355
356
357 File: imagemanagement/step2/internal.py
358 #!/usr/bin/python

```

```

359 import time
360 import subprocess
361
362 def main():
363     command0 = "service glideinwms-pilot status"
364     with open('/tmp/hktesting.log', 'a') as my_log:
365         retcode=subprocess.call(command0, shell=True, stdout=my_log)
366         while retcode != 0:
367             retcode=subprocess.call(command0, shell=True, stdout=my_log)
368             print "retcode = ", retcode
369             time.sleep(1)
370
371     with open('/tmp/hktesting.log', 'a') as my_log:
372         my_log.write("HK glidein running, now trying to stop it")
373
374
375     command1 = "service glideinwms-pilot stop"
376     with open('/tmp/hktesting.log', 'a') as my_log:
377         retcode = subprocess.call(command1, shell=True, stdout=my_log)
378
379
380     with open('/tmp/hktesting.log', 'a') as my_log:
381         retcode=subprocess.call(command0, shell=True, stdout=my_log)
382         print "retcode = ", retcode
383         while retcode != 3:
384             retcode=subprocess.call(command0, shell=True, stdout=my_log)
385             print "retcode = ", retcode
386             time.sleep(1)
387     with open('/tmp/hktesting.log', 'a') as my_log:
388         my_log.write("HK glidein stopped")
389
390     command2 = "cat /dev/null > /var/log/glideinwms-pilot/pilot_launcher.log"
391     command3 = "rm -f /tmp/ephemeral_storage.log"
392     command4 = "umount /home && mv /home.orig/* /home && rmdir /home.orig"
393
394     with open('/tmp/hktesting.log', 'a') as my_log:
395         retcode2 = subprocess.call(command2, shell=True, stdout=my_log)
396         print "retcode 2 = ", retcode2

```

```

397         time.sleep(2)
398         retcode3 = subprocess.call(command3, shell=True, stdout=my_log)
399         print "retcode 3 = ", retcode3
400         time.sleep(2)
401         retcode4 = subprocess.call(command4, shell=True, stdout=my_log)
402         print "retcode 4 = ", retcode4
403         time.sleep(2)
404
405
406     with open('/tmp/hktesting.log', 'a') as my_log:
407         my_log.write( "all jobs done and now shutting down now" )
408
409     command5 = "shutdown -h now"
410     with open('/tmp/hktesting.log', 'a') as my_log:
411         retcode = subprocess.call(command5, shell=True, stdout=my_log)
412
413
414 if __name__ == "__main__":
415     main()
416
417 -----
418 -----
419 -----
420
421
422 File: imagemanagement/step2/internal.sh
423 #!/bin/bash
424 nohup /mnt/internal.py > /dev/null &
425
426 -----
427 -----
428 -----
429
430
431 File: imagemanagement/step2/mount_ephemeral
432 #!/bin/bash
433
434 #

```

```

435 # mount_ephemeral - Attempts to mount ephemeral storage
436 #
437
438
439
440 LOGFILE=/tmp/ephemeral_storage.log
441 VIRTUAL_DISKS="xvdf xvdb vda3 vdb sda2 sdb sdbc"
442
443 echo "Attempting to mount ephemeral storage" | tee $LOGFILE
444
445 if [ ! -d /home/scratchgwms ]; then
446     mkdir -v      /home/scratchgwms 2>&1 | tee --append $LOGFILE
447 fi
448
449 for VD in $VIRTUAL_DISKS ; do
450     echo "Checking /dev/$VD ..." | tee --append $LOGFILE
451     d=`date`
452     fdisk -l /dev/$VD 2>/dev/null | grep Disk >> $LOGFILE
453     if [ $? -eq 0 ]; then
454         echo "Virtual disk seen at /dev/$VD: $d" | tee $LOGFILE
455     fi
456 #HK hack begin
457 string1=`grep scratchgwms /etc/mtab`
458 if [ -n "$string1" ]; then
459     echo "/home/scratchgwms is currently mounted" | tee --append $LOGFILE
460     (cd /home/scratchgwms && rm -rf *) 2>&1 | tee --append $LOGFILE
461     umount      /home/scratchgwms 2>&1 | tee --append $LOGFILE
462 fi
463
464 if [ ! -d /home/scratchgwms ]; then
465     mkdir -v      /home/scratchgwms 2>&1 | tee --append $LOGFILE
466 fi
467
468     mount /dev/$VD /home/scratchgwms 2>&1 | tee --append $LOGFILE
469 #HK hack end
470
471     d=`date`
472     echo "Done with /dev/$VD: $d" | tee --append $LOGFILE

```

```

473         break
474     else
475         echo "No virtual disk seen at /dev/$VD: $d" | tee --append $LOGFILE
476     fi
477 done
478 d=`date`
479 echo "Done: $d" | tee --append $LOGFILE
480
481 -----
482 -----
483 -----
484
485
486 File: imagemanagement/step2/pilot-launcher
487 #!/usr/bin/python
488
489 import os
490 import subprocess
491 import signal
492 import urllib
493 from optparse import OptionParser
494
495 from glideinwms_pilot.errors import PilotError
496 from glideinwms_pilot.errors import TimeoutError
497 from glideinwms_pilot.errors import ConfigError
498 from glideinwms_pilot.errors import ScriptError
499
500 from glideinwms_pilot.vm_utils import chown
501 from glideinwms_pilot.vm_utils import chmod
502 from glideinwms_pilot.vm_utils import cd
503 from glideinwms_pilot.vm_utils import drop_privs
504 from glideinwms_pilot.vm_utils import shutdown_vm
505 from glideinwms_pilot.vm_utils import daemonize
506 from glideinwms_pilot.vm_utils import ls_files_sorted
507 from glideinwms_pilot.vm_utils import sleep
508
509 from glideinwms_pilot.user_data import GlideinWMSUserData
510

```

```

511 from glideinwms_pilot.config import Config
512
513 from glideinwms_pilot.process_handling import execute_cmd
514
515 def retrieve_glidein_startup(config):
516     try:
517         url = "%s/glidein_startup.sh" % config.factory_url
518         script = "%s/glidein_startup.sh" % config.home_dir
519         script, _ = urllib.urlretrieve(url, script)
520     except Exception, ex:
521         raise PilotError("Error retrieving glidein_startup.sh: %s\n" % str(ex))
522
523 def run_scripts(directory, log_writer, max_script_runtime=60, arguments=[]):
524     try:
525         script_list = ls_files_sorted(directory)
526     except Exception, e:
527         message = "An Error has occurred retrieving scripts: %s" % str(e)
528         raise ScriptError(message)
529
530     for script in script_list:
531         try:
532             cmd = "%s/%s" % (directory, script)
533             log_writer.log_info("Executing script %s" % cmd)
534             exit_code = execute_cmd(cmd, max_script_runtime, log_writer,
535                                   arguments, os.environ)
536             log_writer.log_info("Executing script %s ... DONE" % cmd)
537             # have to mod 256 because on some systems, instead of
538             # returning 0 on success, 256 is returned
539             if not int(exit_code) % 256 == 0:
540                 message = "The script (%s) has exited with Exit Code: %s" % (cmd,
541 ... str(exit_code))
542                 log_writer.log_err(message)
543         except Exception, e:
544             message = "An Error has occurred attempting to run script: %s" \
545                 "\n\nError: %s" % (cmd, str(e))
546             log_writer.log_err(message)
547

```

```

548 def main():
549     """
550     Perform all the work necessary to launch a glideinWMS pilot which will
551     attempt to connect back to the user pool.
552
553     1) daemonize this script. This script is lauched via the *nix service
554     mechanisms. We don't want to make it wait forever and we don't
555     want it to be attached to a console.
556     2) Get the user data that was passed to the AMI - Currently it is a
557     tarball.
558     3) untar the tarball. The tarball will contain a proxy, the
559     glidein_startup.sh script and an ini file containing all the extra
560     information needed
561     4) read the ini file
562     5) get the arguments for the glidein_startup.sh script
563     6) create an environment string to pass with final command
564     7) launch the glidein pilot with the appropriate environment
565     """
566
567     usage = "usage: %prog [options] [Site FQDN]"
568     parser = OptionParser(usage=usage)
569     parser.add_option("-d", "--disable-daemon", action="store_true",
570                     dest="disable_daemon", default=False,
571                     help="Disable the daemon functionality and run in "\
572                          "terminal")
573
574     # if the directory, etc/glideinwms, does not exist lets assume that the
575     # config file is in the same directory. We do this now so that this service
576     # can be installed into CVMFS. This is a nasty hack that isn't portable in
577     # the future, but is being done so that we can move forward. We assume
578     # OpenStack since that is the current whim at CERN.
579     init_config_directory = "/etc/glideinwms"
580     if not os.path.exists(init_config_directory):
581         init_config_directory = os.path.dirname(os.path.abspath(__file__))
582     parser.add_option("-c", "--config-file", dest="config_file",
583                     default="%s/glidein-pilot.ini" % init_config_directory,
584                     help="Specify a custom config file")
585

```

```

586 parser.add_option("-p", "--pid-file", dest="pid_file",
587                 default="/tmp/pilot.pid", help="Specify the pidfile")
588
589
590 (options, args) = parser.parse_args()
591
592
593 if options.disable_daemon:
594     print "disable daemon call"
595 else:
596     daemonize(options.pid_file)
597
598 # If config fails, we need to write error to console if available
599 try:
600     config = Config(options.config_file)
601     config.setup()
602     try:
603         # Change to the working directory -- GUARANTEE A KNOWN start dir
604         config.log.log_info('Changing to: %s' % config.home_dir)
605         cd(config.home_dir)
606         config.log.log_info('Now in: %s' % config.home_dir)
607
608         # Run PRE scripts here
609         message = "Running PRE Scripts in %s" % config.pre_script_dir
610         config.log.log_info(message)
611         run_scripts(config.pre_script_dir, config.log,
612                   float(config.max_script_runtime))
613         message = "Running PRE Scripts in ... DONE"
614         config.log.log_info(message)
615
616         # Change to the working directory AGAIN -- IMPORTANT
617         # Needed since PRE scripts may move the home area to a
618         # bigger partition
619         config.log.log_info('Changing to: %s' % config.home_dir)
620         cd(config.home_dir)
621         config.log.log_info('Now in: %s' % config.home_dir)
622
623         # get and extract the user data - should be a tar file

```

```

624         config.log.log_info("Retrieving and extracting user data")
625         userdata = GlideinWMSUserData(config)
626         userdata.extract_user_data()
627
628         # Change the ownership of files in ~glidein_pilot before
629         # dropping privileges
630         chown('%s.%s' % (config.glidein_user, config.glidein_user),
631             config.home_dir)
632 # HK
633         chown('%s.%s' % (config.glidein_user, config.glidein_user),
634             config.scratch_dir)
635 # HK-end
636
637         # drop privileges to the glidein user
638         config.log.log_info("Dropping privs to %s" % config.glidein_user)
639         drop_privs(config.glidein_user)
640
641         # get the glidein_startup.sh script
642         config.log.log_info("Retrieving glidein_startup.sh")
643         retrieve_glidein_startup(config)
644         chmod(0755, "%s/glidein_startup.sh" % config.home_dir)
645
646         # configure pilot launch environment
647         config.log.log_info("Configuring pilot environment...")
648         config.log.log_info("    Username: %s" % config.glidein_user)
649
650         pilot_env = config.get_custom_env()
651         config.log.log_info("    Environment: %s" % str(pilot_env))
652
653         pilot_args = config.pilot_args.split()
654         pilot_args.insert(0, "glidein_startup.sh")
655         config.log.log_info("Pilot arguments: %s" % str(pilot_args))
656
657         # launch the pilot
658         # The pilot will only be allowed to run for config.max_lifetime
659         # seconds before being terminated
660         glidein_startup = "%s/glidein_startup.sh" % config.home_dir
661         config.log.log_info("Launching Pilot (%s)..." % glidein_startup)

```

```

662         _ = execute_cmd(glidein_startup,
663                         float(config.max_lifetime), config.log,
664                         pilot_args, pilot_env)
665     except ScriptError, ex:
666         message = "An Error has occurred: %s" % str(ex)
667         config.log.log_err(message)
668     except PilotError, ex:
669         message = "A PilotError has occurred: %s" % str(ex)
670         config.log.log_err(message)
671     except Exception, ex:
672         config.log.log_err("Error launching pilot: %s" % str(ex))
673     except ConfigError, ex:
674         config.disable_shutdown = False
675
676     try:
677         try:
678             message = "Running Post Scripts in %s" % config.post_script_dir
679             config.log.log_info(message)
680         except:
681             # If a config error occurred originally, then logging isn't
682             # available. This is probably what brought us to this point
683             pass
684
685         # Always run POST Scripts
686         run_scripts(config.post_script_dir, config.log,
687                    float(config.max_script_runtime))
688
689         try:
690             message = "Running Post Scripts ... DONE."
691             config.log.log_info(message)
692         except:
693             # If a config error occurred originally, then logging isn't
694             # available. This is probably what brought us to this point
695             pass
696     except ScriptError, ex:
697         try:
698             message = "An Error has occurred: %s" % str(ex)
699             config.log.log_err(message)

```

```

700     except:
701         # If a config error occurred originally, then logging isn't
702         # available. This is probably what brought us to this point
703         pass
704
705     # No logging is available if the config.setup call errors, so don't try
706     if config.disable_shutdown:
707         print "shutdown disabled"
708     else:
709         ten_minutes = 600 # seconds
710         sleep(ten_minutes)
711         shutdown_vm(options.pid_file)
712
713 if __name__ == "__main__":
714     main()
715
716 -----
717 -----
718 -----
719
720
721 File: imagemanagement/step2/README
722 in fclheadgpvm01.fnal.gov, the following is there..
723
724 /etc/cron.d/step2_imaging
725 0 7 * * 1-5 root /opt/gcso/opennebula/imaging/step2_imaging_envron.sh
726
727 /opt/gcso/opennebula/imaging/step2_imaging_envron.sh
728
729
730 /opt/gcso/opennebula/imaging/step2_imaging_external.py
731
732 onetemplate list oneadmin
733 185 oneadmin          oneadmin          make_prvm          09/02 12:53:39
734
735 /cloud/images/OpenNebula/scripts/one4.x/contextualization/hkilot.sh
736 /cloud/images/OpenNebula/scripts/one4.x/contextualization/internal.sh
737 /cloud/images/OpenNebula/scripts/one4.x/contextualization/internal.py

```

```

738
739 /cloud/images/OpenNebula/scripts/one4.x/contextualization/pilot-launcher
740 /cloud/images/OpenNebula/scripts/one4.x/contextualization/mount_ephemeral
741 /cloud/images/OpenNebula/scripts/one4.x/contextualization/config.py
742 The above 3 files will not be needed when I have closed the ticket 10389 to generate a new
... glideinwms-vm-core 1.0.6 RPM
743
744 INIT_SCRIPTS="init.sh credentials.sh kerberos.sh hkpilot.sh"
745
746 IMAGE="SLF6Vanilla",
747
748 oneimage list oneadmin
749   ID USER      GROUP      NAME           DATASTORE     SIZE TYPE PER STAT RVMS
750   4 oneadmin  oneadmin   SLF6Vanilla    cloud_imag     256G OS   No used 197
751
752
753
754 /cloud/images/OpenNebula/scripts/one4.x/contextualization/hkpilot.sh is the puppet apply
755
756   exec { 'hktempoverwrite':
757     command => '
758 /bin/mount -t iso9660 /dev/sr0 /mnt;
759 /bin/cp -f /mnt/pilot-launcher /usr/sbin/pilot-launcher;
760 /bin/cp -f /mnt/mount_ephemeral /usr/libexec/glideinwms_pilot/PRE/mount_ephemeral;
761 /bin/cp -f /mnt/config.py /usr/lib/python2.6/site-packages/glideinwms_pilot/config.py
762 ',
763
764
765
766
767
768 /cloud/images/OpenNebula/scripts/one4.x/contextualization/glidein_startup.sh is not needed
... any more.
769
770
771
772
773

```

```

774
775 -----
776 -----
777 -----
778
779
780 File: imagemanagement/step2/step2_imaging_external.py
781 #!/usr/bin/env python
782 import string
783 import subprocess
784 import logging
785 import textwrap
786 import sys
787 import socket
788 import datetime
789 import time
790 import os
791 import time
792 try:
793     import xml.etree.cElementTree as ET
794 except:
795     import xml.etree.ElementTree as ET
796
797
798 def main():
799     my_env = os.environ.copy()
800
801     logging.basicConfig(level=logging.DEBUG,
802                         format='%(asctime)s %(levelname)-8s %(message)s',
803                         datefmt='%b %d %H:%M:%S',
804                         filename='/tmp/step2_image_convert.log')
805
806 #####
807 #####
808     subprocess.Popen( ["onetemplate instantiate make_prvm --name 'step2-AWS' "], env=my_env,
... stdout=subprocess.PIPE, shell=True )
809
810 # need to pause a bit here after onetemplate instantiate before we will be able to do onevm

```

```

810.. show command
811     time.sleep(60)
812
813
814     getxml = subprocess.Popen( ["onevm show 'step2-AWS' --xml"], env=my_env,
... stdout=subprocess.PIPE, shell=True )
815     std_out_xml, std_err_xml = getxml.communicate()
816     root = ET.fromstring( std_out_xml )
817     vm_id      = root.find('ID').text
818     vm_state_text = root.find('STATE').text
819     vm_state_int  = int(vm_state_text)
820
821     logging.info('vm state  = %s' % vm_state_text)
822     logging.info('vm id    = %s' % vm_id)
823
824     while vm_state_int != 3:
825         logging.info('vm is not running yet, sleep 10 more seconds')
826         time.sleep(5)
827
828         getxml = subprocess.Popen(["onevm show 'step2-AWS' --xml"], env=my_env,
... stdout=subprocess.PIPE, shell=True)
829         std_out_xml, std_err_xml = getxml.communicate()
830         root = ET.fromstring( std_out_xml )
831         vm_state_text = root.find('STATE').text
832         vm_state_int  = int(vm_state_text)
833         logging.info('vm state  = %s' % vm_state_text)
834
835         bare_metal_host = root.find('HISTORY_RECORDS/HISTORY/HOSTNAME')
836         logging.info('VM is now running and its bare metal = %s' % bare_metal_host.text)
837
838 #HK> new addition
839     kvcommand = '/cloud/images/OpenNebula/scripts/one4.x/onehostname {vmid}'.format(
... vmid=vm_id )
840     getip = subprocess.Popen( kvcommand, env=my_env, stdout=subprocess.PIPE, shell=True)
841     std_out_ip, std_err_ip = getip.communicate()
842     logging.info( 'vm ip address = %s' % std_out_ip.rstrip('\n') )
843
844     kvcommand2 = 'ssh -l root {vmip} uname -a'.format( vmip=std_out_ip.rstrip('\n') )

```

```

845
846     getkv = subprocess.Popen( kvcommand2, env=my_env, stdout=subprocess.PIPE, shell=True)
847     std_out_pl, std_err_pl = getkv.communicate()
848     pilotresult = std_out_pl.rstrip('\n')
849     logging.info( 'pilot result = %s' % pilotresult )
850
851     while not pilotresult.startswith('Linux'):
852         logging.info('vm does not have network, sleep 10 more seconds')
853         time.sleep(10)
854         getkv = subprocess.Popen( kvcommand2, env=my_env, stdout=subprocess.PIPE, shell=True)
855         std_out_pl, std_err_pl = getkv.communicate()
856         pilotresult = std_out_pl.rstrip('\n')
857         logging.info( 'result inside while = %s' % pilotresult )
858
859     logging.info('Finally vm does have network, now we are getting the kernel version')
860
861     kvcommand3 = 'ssh -l root {vmip} uname -r'.format( vmip=std_out_ip.rstrip('\n') )
862     getkv = subprocess.Popen( kvcommand3, env=my_env, stdout=subprocess.PIPE, shell=True)
863     std_out_kv, std_err_kv = getkv.communicate()
864     kernelresult = std_out_kv.rstrip('\n')
865     logging.info('Kernel result = %s' % kernelresult)
866
867     with open('/tmp/kernelversion.txt', 'w') as hkmylog:
868         hkmylog.write( kernelresult )
869 #HK> new addition
870
871     logging.info('vm is running and now we are waiting for it to go to pooff by glideinwms
... timeout mechanism')
872 ## the internal script will initiate the shutdown
873 # now we wait until the vm does shut down which is executed by the second internal script
874
875     while vm_state_int != 8:
876         logging.info("the vm is still not down")
877         time.sleep(60)
878
879         getxml = subprocess.Popen(["onevm show 'step2-AWS' --xml"], env=my_env,
... stdout=subprocess.PIPE, shell=True)
880         std_out_xml, std_err_xml = getxml.communicate()

```

```

881     root = ET.fromstring( std_out_xml )
882     vm_state_text = root.find('STATE').text
883     vm_state_int = int(vm_state_text)
884
885
886     logging.info('vm is down now, we are going to copy the image out')
887
888 # now the VM is shutdown by the internal script,
889 # this means the opennebula did not kill the VM, i.e. the VMHost still has the disk.0 file at
... VMHost:/var/lib/one/datastores/100/VMID/
890
891 # Now, we copy the disk.0 This is a potential risk, i.e. when /opt area does not have
... sufficient space, this script will fail to copy out disk.0 from VMHost.
892 # tmp_destination = "/cloud/images/hyunwoo/hkdisk.0"
893     tmp_destination = "/opt/gcso/opennebula/imaging/tmp_step2.img"
894     if os.path.exists(tmp_destination):
895         os.remove(tmp_destination)
896         logging.info('%s is deleted' % tmp_destination)
897
898     scp_value = "scp oneadmin@{hostname}:/var/lib/one/datastores/104/{vmid}/disk.0
... {tmpimage}".format(tmpimage=tmp_destination, hostname=bare_metal_host.text, vmid=vm_id)
899
900     subprocess.check_call(scp_value, shell=True)
901
902     logging.info('copy the image out is complete')
903     time.sleep(10)
904
905
906 ## We need to find out the image location
907 ## Now, we need to replace the old image with the new image:
908     getxml = subprocess.Popen( ["oneimage show 'SLF6_prvm' --xml"], env=my_env,
... stdout=subprocess.PIPE, shell=True )
909     std_out_xml, std_err_xml = getxml.communicate()
910     root = ET.fromstring( std_out_xml )
911     image_destination = root.find('SOURCE').text
912
913     get_date = datetime.date.today()
914     mv_command = ("mv {tmpimage} {image}.new &&"

```

```

915         " mv -f {image} {image}.{today} && "
916         " mv {image}.new {image} && chown oneadmin:oneadmin "
917         "{image} && chmod 660 {image} ").format (tmpimage=tmp_destination,
... image=image_destination, today=str(get_date) )
918
919     subprocess.check_call(mv_command, shell=True)
920
921     logging.info('copy the image in the image repository is complete')
922
923 #HK, can we delete the poff VM at the end of this script?
924     subprocess.Popen( ["onevm delete 'step2-AWS' "], env=my_env, stdin=subprocess.PIPE,
... stdout=subprocess.PIPE, shell=True )
925
926     logging.info('onevm delete step2-AWS is complete')
927
928 if __name__ == "__main__":
929     main()
930
931
932 -----
933 -----
934 -----
935
936
937 File: imagemanagement/step3/cloud.cfg
938 #Bare-bone cloud.cfg, add parameters as needed for FermiCloud
939
940 user: root
941
942 #If this is not explicitly false, cloud-init will change things so that root
943 #login via ssh is disabled. Set it false to allow root login via ssh keypair.
944
945 disable_root: false
946
947 #add additional cloud-init output logging
948
949 output: {all: '| tee -a /var/log/cloud-init-output.log'}
950

```

```

951 #Since cloud-init runs at multiple stages of boot, this needs to be set so
952 #it can log in all of them to /var/log/cloud-init.
953
954 syslog_fix_perms: null
955
956 #This is the piece that makes userdata work. You need this to have userdata
957 #scripts be run by cloud-init.
958
959 datasource_list: [Ec2]
960 datasource:
961   Ec2:
962     metadata_urls: ['http://169.254.169.254']
963
964 #modules that run early in boot
965
966 cloud_init_modules:
967 - bootcmd #for running commands during boot. Commands can be defined in cloud-config
  userdata.
968
969 #modules that run after boot
970
971 cloud_config_modules:
972 - runcmd #like bootcmd, but runs after boot. Use this instead of bootcmd for after boot
  processing.
973
974 #modules that run at some point after config is finished
975
976 cloud_final_modules:
977 - scripts-per-once #all of these run scripts at specific events. Like bootcmd, can be
  defined in cloud-config.
978 - scripts-per-boot
979 - scripts-per-instance
980 - scripts-user
981 - phone-home #if defined, can make a post request to a specified url when done booting
982 - final-message #if defined, can write a specified message to the log
983 - power-state-change #if defined, can trigger stuff based on power state changes
984
985 system_info:

```

```

986   distro: rhel
987
988 # vim:syntax=yaml
989 -----
990 -----
991 -----
992
993
994 File: imagemanagement/step3/cms.cern.ch.local
995 export CMS_LOCAL_SITE="/etc/cvmfs/SITECONF/T3_US_HEP_Cloud"
996 -----
997 -----
998 -----
999
1000
1001 File: imagemanagement/step3/Convert-boto.py
1002 #!/usr/bin/env python
1003 # encoding: utf-8
1004 from hepcloud_imaging_boto import hepcloud_upload
1005
1006 import sys
1007 import os
1008 import subprocess
1009 import logging
1010 import textwrap
1011 import datetime
1012 import time
1013
1014 from argparse import ArgumentParser
1015 from argparse import RawDescriptionHelpFormatter
1016
1017 __all__ = []
1018 __version__ = 0.1
1019 __date__ = '2014-07-15'
1020 __updated__ = '2014-07-15'
1021
1022 DEBUG = 1
1023 TESTRUN = 0

```

```

1024 PROFILE = 0
1025
1026 class CLIError(Exception):
1027     '''Generic exception to raise and log different fatal errors.'''
1028     def __init__(self, msg):
1029         super(CLIError).__init__(type(self))
1030         self.msg = "E: %s" % msg
1031     def __str__(self):
1032         return self.msg
1033     def __unicode__(self):
1034         return self.msg
1035
1036 class Timer:
1037     def __enter__(self):
1038         self.start = time.time()
1039         return self
1040
1041     def __exit__(self, *args):
1042         self.end = time.time()
1043         self.interval = self.end - self.start
1044
1045 def get_help():
1046     print "dummy help"
1047
1048 def copy_to_image_location(vm_script_location, vm_name, vm_image_location):
1049     """ This function copies the selected Fermicloud VM image to a worker VM image.
1050     """
1051     get_date = datetime.date.today()
1052     time_differential = datetime.timedelta(days=7)
1053     delete_date = str(get_date - time_differential)
1054     logging.info("Start: Copying the selected Fermicloud VM image. Function
... copy_to_image_location.")
1055     """ Change to location of your scripts below.
1056     """
1057     cp_command = (
1058         "cp -f {v_script_location}/Fermi_AWS_Modifications.sh /data"
1059         " && cp -f {v_script_location}/Fermi_AWS_Resize.sh /data"
1060         " && cp -f {v_script_location}/hepcloud-init-workernode /data" # new by HK

```

```

1061         " && cp -f {v_script_location}/cms.cern.ch.local /data" # new by HK
1062         " && cp -f {v_script_location}/site-local-config.xml /data" # new by HK
1063         " && cp -f {v_script_location}/storage.xml /data" # new by HK
1064         " && cp -f {v_script_location}/ec2-get-ssh /data"
1065         " && cp -f {v_script_location}/cloud.cfg /data"
1066         " && kinit -k -t /var/adm/krb5/cloudadminpp.keytab
... cloudadmin/cron/fermicloudpp.fnal.gov@FNAL.GOV"
1067         " && scp {v_image_location} /data/{v_name}.qcow2tmp"
1068         ).format (today=str(get_date), v_name=vm_name,
... v_script_location=vm_script_location, v_image_location=vm_image_location, remove=delete_date)
1069     with open('/opt/gcso/awsexport/aws_image_convert.log', 'a') as my_log:
1070
1071     try:
1072         with Timer() as t:
1073             subprocess.check_call(cp_command, shell=True, stdout=my_log, stderr=my_log)
1074             logging.info("Stop: Completed Copying the selected Fermicloud VM image.
... Function copy_to_image_location.")
1075     except:
1076         logging.error("Couldn't copy image to temp area")
1077         raise Exception("Couldn't copy image to temp area! Aborting.")
1078         sys.exit(1)
1079     finally:
1080         min_interval = t.interval / 60
1081         print('Copying took %.03f minutes.' % min_interval)
1082
1083 def resize_image(vm_name):
1084     """ This function resizes the worker Fermicloud VM image for AWS image import.
1085     """
1086     get_date = datetime.date.today()
1087     logging.info("Start: Resizing the worker Fermicloud VM image. Function resize_image.")
1088     resize_command = (
1089         "cd /data"
1090         " && ./Fermi_AWS_Resize.sh {v_name}"
1091         " && rm -f /data/Fermi_AWS_Resize.sh"
1092         ).format (today=str(get_date), v_name=vm_name)
1093     with open('/opt/gcso/awsexport/aws_image_convert.log', 'a') as my_log:
1094     try:
1095         with Timer() as t:

```

```

1096         subprocess.check_call(resize_command, shell=True, stdout=my_log,
...     stderr=my_log)
1097         logging.info("Stop: Completed Resizing the worker Fermicloud VM image.
... Function resize_image.")
1098     except:
1099         logging.error("Couldn't resize image in temp area")
1100         raise Exception("Couldn't resize image in temp area! Aborting.")
1101         sys.exit(1)
1102     finally:
1103         min_interval = t.interval / 60
1104         print('Resizing took %.03f minutes.' % min_interval)
1105
1106 def convert_image(vm_name, kernel_ver, eph_mount):
1107     """ This function converts the worker Fermicloud VM image for AWS specifics.
1108     """
1109     get_date = datetime.date.today()
1110     logging.info("Start: Converting the worker Fermicloud VM image. Function convert_image.")
1111     convert_command = (
1112         "mkdir -p /data/work"
1113         " && guestmount -a /data/{v_name}.raw -m /dev/sda1 /data/work"
1114         " && mv /data/Fermi_AWS_Modifications.sh /data/work"
1115         " && mv /data/hepcloud-init-workernode /data/work" # new by HK
1116         " && mv /data/cms.cern.ch.local /data/work" # new by HK
1117         " && mv /data/site-local-config.xml /data/work" # new by HK
1118         " && mv /data/storage.xml /data/work" # new by HK
1119         " && mv /data/ec2-get-ssh /data/work"
1120         " && mv /data/cloud.cfg /data/work"
1121         " && /usr/sbin/chroot /data/work ./Fermi_AWS_Modifications.sh
... {v_kernel_ver} {v_eph_mount}"
1122         " && sleep 10"
1123         " && rm -f /data/work/Fermi_AWS_Modifications.sh && rm -f
... /data/work/ec2-get-ssh && rm -f /data/work/cloud.cfg"
1124         " && rm -f /data/Fermi_AWS_Modifications.sh && rm -f /data/ec2-get-ssh
... && rm -f /data/cloud.cfg"
1125         " && fusermount -uz /data/work && rmdir /data/work"
1126         ).format(today=str(get_date), v_name=vm_name, v_kernel_ver=kernel_ver,
... v_eph_mount=eph_mount)
1127     with open('/opt/gcso/awsexport/aws_image_convert.log', 'a') as my_log:

```

```

1128
1129     try:
1130         with Timer() as t:
1131             subprocess.check_call(convert_command, shell=True, stdout=my_log,
... stderr=my_log)
1132             logging.info("Stop: Completed Converting the worker Fermicloud VM image.
... Function convert_image.")
1133     except:
1134         logging.error("Couldn't convert image in temp area")
1135         raise Exception("Couldn't convert image in temp area! Aborting.")
1136         sys.exit(1)
1137     finally:
1138         min_interval = t.interval / 60
1139         print('Converting took %.03f minutes.' % min_interval)
1140
1141 def import_image(vm_name, aws_image_name):
1142     get_date = datetime.date.today()
1143     logging.info("Start: Importing the worker Fermicloud VM image. Function import_image.")
1144     with open('/opt/gcso/awsexport/aws_image_convert.log', 'a') as my_log:
1145         try:
1146             with Timer() as t:
1147                 image_name = "/data/" + vm_name + ".raw"
1148                 print image_name
1149                 hepcloud_upload(image_name)
1150         except:
1151             logging.error("Couldn't import image to AWS")
1152             raise Exception("Couldn't import image to AWS! Aborting.")
1153             sys.exit(1)
1154         finally:
1155             min_interval = t.interval / 60
1156             print('Importing took %.03f minutes.' % min_interval)
1157
1158 def main(argv=None):
1159     """ Change location of Log file.
1160     """
1161     logging.basicConfig(level=logging.DEBUG,
1162                         format='%%(asctime)s %(levelname)-8s %(message)s',
1163                         datefmt='%b %d %H:%M:%S',

```

```

1164         filename='/opt/gcso/awsexport/aws_image_convert.log')
1165
1166     my_env = os.environ.copy()
1167
1168     '''Command line options.'''
1169
1170     if argv is None:
1171         argv = sys.argv
1172     else:
1173         sys.argv.extend(argv)
1174
1175     get_help()
1176     program_name = os.path.basename(sys.argv[0])
1177     program_version = "v%s" % __version__
1178     program_build_date = str(__updated__)
1179     program_version_message = '%%(prog)s %s (%s)' % (program_version, program_build_date)
1180     try:
1181         with Timer() as t:
1182             # Setup argument parser
1183             parser = ArgumentParser(description="dummy licence",
1184 ... formatter_class=RawDescriptionHelpFormatter)
1185             parser.add_argument('-V', '--version', action='version',
1186 ... version=program_version_message)
1187             parser.add_argument(dest="cvm_script_location", help="Location of all scripts",
1188 ... metavar="cvm_script_location")
1189             parser.add_argument(dest="cvm_image_location", help="Location of Fermi Image",
1190 ... metavar="cvm_image_location")
1191             parser.add_argument(dest="ckernel_ver", help="Kernel version of Fermi Image",
1192 ... metavar="ckernel_ver")
1193             parser.add_argument(dest="cvm_name", help="Fermicloud VM name",
1194 ... metavar="cvm_name")
1195             parser.add_argument(dest="caws_image_name", help="AWS AMI VM name",
1196 ... metavar="caws_image_name")
1197             parser.add_argument(dest="caws_eph_mount", help="AWS ephemeral mount dir
1198 ... (/ephemeral_mount_dir or none)", metavar="caws_eph_mount")
1199
1200         # Process arguments
1201         args = parser.parse_args()

```

```

1202     cvm_script_location = args.cvm_script_location
1203     cvm_image_location = args.cvm_image_location
1204     ckernel_ver = args.ckernel_ver
1205     cvm_name = args.cvm_name
1206     caws_image_name = args.caws_image_name
1207     caws_eph_mount = args.caws_eph_mount
1208
1209     logging.info('Begin script run')
1210     print("Arguments supplied:")
1211     print("location of all files and scripts->", cvm_script_location)
1212     print("location of fermicloud vm image->", cvm_image_location)
1213     print("kernel version of fermicloud vm image->", ckernel_ver)
1214     print("fermicloud vm image name->", cvm_name)
1215     print("aws vm image name->", caws_image_name)
1216     print("aws ephemeral mount->", caws_eph_mount)
1217
1218     print("Starting AWS VM conversion. This job can take up to 60 minutes for PV and
1219 ... 84 for HVM...")
1220     print("Copying the Golden Fermicloud VM image takes under 1 minute...")
1221     copy_to_image_location(cvm_script_location, cvm_name, cvm_image_location)
1222     print("Resizing the worker Fermicloud VM image takes up to 20 minutes for PV and
1223 ... 10 for HVM...")
1224     resize_image(cvm_name)
1225     print("Converting the worker Fermicloud VM image takes over 24 minutes...")
1226     convert_image(cvm_name, ckernel_ver, caws_eph_mount)
1227     print("Importing the raw image to AWS takes up to 15 minutes for PV and 49 for
1228 ... HVM...")
1229     import_image(cvm_name, caws_image_name)
1230
1231     print("Completed AWS VM conversion.")
1232     logging.info('End script run')
1233     return 0
1234
1235     except KeyboardInterrupt:
1236         return 0
1237     except Exception, e:
1238         if DEBUG or TESTRUN:
1239             raise(e)

```

```

1229         indent = len(program_name) * " "
1230         sys.stderr.write(program_name + ": " + repr(e) + "\n")
1231         sys.stderr.write(indent + " for help use --help")
1232         return 2
1233     finally:
1234         min_interval = t.interval / 60
1235         print('Job took %.03f minutes. Thank you.' % min_interval)
1236
1237
1238 if __name__ == "__main__":
1239     sys.exit(main())
1240
1241 -----
1242 -----
1243 -----
1244
1245
1246 File: imagemanagement/step3/ec2-get-ssh
1247 #!/bin/bash
1248 # chkconfig: 2345 95 20
1249 # processname: ec2-get-ssh
1250 # description: Capture AWS public key credentials for EC2 user
1251
1252 # Source function library
1253 . /etc/rc.d/init.d/functions
1254
1255 # Source networking configuration
1256 [ -r /etc/sysconfig/network ] && . /etc/sysconfig/network
1257
1258 # Replace the following environment variables for your system
1259 export PATH=/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin
1260
1261 # Check that networking is configured
1262 if [ "${NETWORKING}" = "no" ]; then
1263     echo "Networking is not configured."
1264     exit 1
1265 fi
1266

```

```

1267 start() {
1268     if [ ! -d /root/.ssh ]; then
1269         mkdir -p /root/.ssh
1270         chmod 700 /root/.ssh
1271     fi
1272     # Retrieve public key from metadata server using HTTP
1273     curl -f http://169.254.169.254/latest/meta-data/public-keys/0/openssh-key >
... /tmp/my-public-key
1274     if [ $? -eq 0 ]; then
1275         echo "EC2: Retrieve public key from metadata server using HTTP."
1276         cat /tmp/my-public-key >> /root/.ssh/authorized_keys
1277         chmod 600 /root/.ssh/authorized_keys
1278         rm /tmp/my-public-key
1279     fi
1280 }
1281
1282 stop() {
1283     echo "Nothing to do here"
1284 }
1285
1286 restart() {
1287     stop
1288     start
1289 }
1290
1291 # See how we were called.
1292 case "$1" in
1293     start)
1294         start
1295         ;;
1296     stop)
1297         stop
1298         ;;
1299     restart)
1300         restart
1301         ;;
1302     *)
1303         echo $"Usage: $0 {start|stop|restart}"

```

```

1304     exit 1
1305 esac
1306
1307 exit $?
1308 -----
1309 -----
1310 -----
1311
1312
1313 File: imagemanagement/step3/Fermi_AWS_Modifications.sh
1314 #!/bin/bash
1315 # This is a script to add or modify several OS networking files required for the AWS image
... conversion
1316
1317 # Remove glideinwms-vm-one (OpenNebula), if exists, and Install glideinwms-vm-ec2 (AWS)
1318
1319 yum -y remove glideinwms-vm-one
1320 #yum -y install glideinwms-vm-core # removed by HK because we have not created a new RPM, but
... rather we are still overwriting 3 files after rpm install in the previous stage..
1321 yum -y install glideinwms-vm-ec2
1322
1323 # Install cloud-init for aws metadata user data and scripts
1324
1325 rpm -Uvh http://download.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
1326 yum -y install cloud-init cloud-utils
1327
1328 #HK> python-backports-1.0-4.el6.x86_64.rpm is downloaded as dependency for cloud-init
1329 #HK> the following URL does not exist
1330 #HK> yum -y install
... http://repos.fedorapeople.org/repos/openstack/openstack-havana/epel-6/python-backports-1.0-4.
... el6.x86_64.rpm
1331
1332 yum -y install python-pip
1333 #pip install awscli
1334
1335
1336 cd /
1337 mv -f cloud.cfg /etc/cloud

```

```

1338
1339 # Modify ifcfg-eth0
1340 cd /etc/sysconfig/network-scripts
1341 rm -f ifcfg-eth0
1342 echo DEVICE=eth0 >> ifcfg-eth0
1343 echo BOOTPROTO=dhcp >> ifcfg-eth0
1344 echo ONBOOT=yes >> ifcfg-eth0
1345 echo TYPE=Ethernet >> ifcfg-eth0
1346
1347 # Modify network
1348 cd /etc/sysconfig
1349 rm -f network
1350 echo NETWORKING=yes >> network
1351
1352 # Remove fermi network resolv and hosts that AWS will not use
1353 cd /etc
1354 rm -f /etc/resolv.conf
1355 rm -f /etc/hosts
1356 rm -f /etc/hosts.allow
1357 rm -f /etc/hosts.deny
1358
1359 echo " " >> /etc/hosts # added by HK/ST
1360
1361 # Remove fermi specifics that AWS will not use
1362 rm -f /etc/yp.conf
1363 # These are no longer in the golden image by default
1364 # Instead we have to actually make an auto.master
1365
1366 cat > /etc/auto.master << EOF
1367 /misc /etc/auto.misc
1368 /net -hosts
1369 +auto.master
1370 /cvmfs /etc/auto.cvmfs
1371 EOF
1372
1373 #rm -f /etc/auto.master
1374 #rm -f /etc/auto.misc
1375 #these 5 files are no longer used,

```

```

1376 # the script name is vmcontext now but there is no
1377 # reason to remove it because
1378 # without opennebula style user data it will
1379 # just not do anything
1380 #rm -f /etc/rc.d/rc3.d/K99.credentials
1381 #rm -f /etc/rc.d/rc3.d/S09one-context
1382 #rm -f /etc/rc.d/rc3.d/K84one-context
1383 #rm -f /etc/init.d/one-context
1384 #rm -f /etc/init.d/.credentials*
1385
1386 # Create grub.conf for AWS paravirtual
1387 cd /boot/grub
1388 rm -f grub.conf
1389 echo default=0 >> grub.conf
1390 echo timeout=0 >> grub.conf
1391 echo 'title Scientific Linux Fermi ('$1')' >> grub.conf
1392
1393 echo 'root (hd0,0)' >> grub.conf
1394 echo 'kernel /boot/vmlinuz-$$$1' ro root=/dev/xvda1 rd_NO_PLYMOUTH' >> grub.conf
1395
1396 echo 'initrd /boot/initramfs-$$$1.img' >> grub.conf
1397
1398 # Create symbolic link for boot
1399 cd /boot; ln -s . boot
1400
1401 # Create fstab for AWS paravirtual
1402 cd /etc
1403 rm -f fstab
1404
1405
1406 echo '/dev/xvda1          /                    ext3    defaults        1 1' >> fstab
1407 # Kirk original version
1408 #HK experiment $2 should be eph_mount set to /var/lib/cvmfs2
1409 #HK> we are hardwiring /dev/xvdc which is based on how Fermi_AWS_Import.sh is using -b option
1410 ... to register_image
1411 if [ "$2" != "none" ]; then
1412     mkdir -p $2
1413     echo 'CVMFS_CACHE_BASE= '$2'' >>/etc/cvmfs/default.local

```

```

1413     echo '/dev/xvdc          '$2'                ext3    defaults        0 0' >> fstab
1414 fi
1415
1416 echo 'tmpfs                /dev/shm          tmpfs    defaults        0 0' >> fstab
1417 echo 'devpts                /dev/pts          devpts   gid=5,mode=620 0 0' >> fstab
1418 echo 'sysfs                  /sys              sysfs    defaults        0 0' >> fstab
1419 echo 'proc                    /proc             proc     defaults        0 0' >> fstab
1420
1421 # Modify rc.local final init script to add additional ephemeral drive and mount scratch there
1422 #HK deleted completely in order to clear any confusion
1423
1424 # Modify sshd_config
1425 cd /etc/ssh
1426 rm -f sshd_config
1427 echo 'SyslogFacility AUTHPRIV' >> sshd_config
1428 echo 'RSAAuthentication no' >> sshd_config
1429 echo 'PubkeyAuthentication yes' >> sshd_config
1430 echo 'AuthorizedKeysFile .ssh/authorized_keys' >> sshd_config
1431 echo 'PasswordAuthentication yes' >> sshd_config
1432 echo 'KerberosAuthentication no' >> sshd_config
1433 echo 'KerberosOrLocalPasswd no' >> sshd_config
1434 echo 'KerberosTicketCleanup no' >> sshd_config
1435 echo 'GSSAPIAuthentication no' >> sshd_config
1436 echo 'GSSAPICleanupCredentials no' >> sshd_config
1437 echo 'UsePAM no' >> sshd_config
1438 echo 'AllowTcpForwarding yes' >> sshd_config
1439 echo 'X11Forwarding yes' >> sshd_config
1440 echo 'UseLogin no' >> sshd_config
1441 echo 'UseDNS no' >> sshd_config
1442
1443 # new by HK
1444 cd /
1445 # new by ST/HK
1446 mv hepcloud-init-workernode /etc/rc.d/init.d/ # replacing S99local as ST wanted
1447 /bin/chmod 755 /etc/rc.d/init.d/hepcloud-init-workernode
1448
1449 # for Frontier
1450 mkdir -p /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/

```

```

1451 mkdir -p /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/PhEEx/
1452 mv site-local-config.xml /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/
1453 mv storage.xml /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/PhEEx/
1454
1455 # to set the correct squid server for /etc/cvmfs/default.local
1456 mkdir -p /etc/cvmfs/config.d/
1457 mv cms.cern.ch.local /etc/cvmfs/config.d/
1458 /sbin/chkconfig --add hepccloud-init-workernode
1459 # end new by ST/HK
1460
1461 # modified on Sep 11 2015
1462 rm -f /etc/grid-security/certificates/*.r0
1463 /sbin/chkconfig fetch-crl-cron off
1464 /sbin/chkconfig fetch-crl-boot off
1465
1466
1467 # Create ec2-get-ssh authentication script for public keypair
1468 cd /
1469 mv ec2-get-ssh /etc/init.d
1470 /bin/chmod +x /etc/init.d/ec2-get-ssh
1471
1472 # Modify /sbin/chkconfig services
1473 /sbin/chkconfig ec2-get-ssh on
1474 /sbin/chkconfig rpcbind off
1475 /sbin/chkconfig postfix off
1476 # /sbin/chkconfig autofs off
1477 /sbin/chkconfig vmcontext off
1478 # Prevent 10 minute automatic vm shutdown for testing (**turn back on after testing**)
1479 # /sbin/chkconfig glideinwms-pilot off
1480
1481 # Clear history
1482 history -c
1483
1484 cat /etc/cvmfs/default.local
1485
1486 # exit chroot
1487 exit
1488

```

```

1489 -----
1490 -----
1491 -----
1492
1493
1494 File: imagemanagement/step3/Fermi_AWS_Resize.sh
1495 #!/bin/bash
1496 # This is a script to resize the converted image (accepts param $1=vmimage name $2="hvm") from
... a QCOW2 256G image down to a 12G primary partition required for the AWS HVM image upload or a
... 3G primary partiton for a AWS PV image upload
1497
1498 # Remove previous raw image, if exists, and formats new raw image
1499 cd /data
1500 rm -f $1.raw
1501
1502 # Use guestfish to delete 2nd and 3rd partitions, if present, clean up count errors and resize
... primary partition content (about 8.5 minutes)
1503 guestfish -a $1.qcow2tmp <<_EOF1_
1504 run
1505 part-del /dev/sda 2
1506 part-del /dev/sda 3
1507 _EOF1_
1508 sleep 2
1509 guestfish -a $1.qcow2tmp <<_EOF2_
1510 run
1511 e2fsck-f /dev/sda1
1512 _EOF2_
1513 sleep 2
1514
1515 guestfish -a $1.qcow2tmp <<_EOF3_
1516 run
1517 resize2fs-size /dev/sda1 6G
1518 _EOF3_
1519 sleep 2
1520 # Resize boot partiton takes about 3.5 minutes
1521 qemu-img create -f raw $1.raw 6150M
1522 virt-resize --resize /dev/sda1=6G $1.qcow2tmp $1.raw
1523

```

```

1524 # Status of converted raw file
1525 qemu-img info $1.raw
1526 -----
1527 -----
1528 -----
1529
1530
1531 File: imagemanagement/step3/hepcloud-init-workernode
1532 #!/bin/sh
1533 ### BEGIN INIT INFO
1534 # chkconfig: 2345 27 25
1535 # Provides:          hepcloud-init-workernode
1536 # Required-Start:   $local_fs $network
1537 # Should-Start:     $time
1538 # Required-Stop:
1539 # Should-Stop:
1540 # Default-Start:    2 3 4 5
1541 # Default-Stop:     0 1 6
1542 # Short-Description: Fix host name and az-specific config files
1543 # Description:       Start cloud-init and runs the initialization phase
1544 #                    and any associated initial modules as desired.
1545 #test
1546 ### END INIT INFO
1547 LOG="/var/log/hepcloud-init-workernode.log"
1548 RETVAL=0
1549
1550 prog="hepcloud-init-workernode"
1551
1552 start() {
1553
1554 touch /var/lock/subsys/hepcloud-init-workernode
1555 #
1556 # get the public hostname of the EC2 instance and change the
1557 # output of the hostname command to match that. (needed for gridftp).
1558 #
1559 mypublicip=`GET http://169.254.169.254/latest/meta-data/public-ipv4`
1560 myrc=$?
1561 if [ $myrc -ne 0 ]

```

```

1562 then
1563     echo "My public IP not defined" >> $LOG
1564     return 6
1565 fi
1566 nslookup $mypublicip | grep "name =" | awk -F ' ' '{print $4}' | sed 's/com\.\/com/' >
... /etc/hostname
1567 myrc=$?
1568 if [ $myrc -ne 0 ]
1569 then
1570     echo "My public DNS name not defined" >> $LOG
1571     return 7
1572 fi
1573 hostname -F /etc/hostname
1574
1575 # This script modifies the CVMFS and Frontier scripts on VM startup
1576 # to point to the ELB-enabled squid stack for the respective
1577 # availability zone
1578
1579 zone=$(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone)
1580 echo $zone
1581 echo $zone >> $LOG
1582 addr="http://elb2.$zone.elb.fnaldata.org:3128" #HK> export http_proxy does not like two
... back-slashes, I had to remove them
1583
1584 # new code by ST and HK #####
1585 export http_proxy=$addr
1586 wget http://cvmfs.fnal.gov:8000/cvmfs/cms.cern.ch/.cvmfspublished
1587 returnvalue=$?
1588 if [ $returnvalue -ne 0 ]
1589 then
1590     echo "Squid Server is not accessible in $zone, trying us-west-2b" >> $LOG
1591
1592     if [ $zone != "us-west-2b" ]
1593     then
1594         uswest2baddr="http://elb2.us-west-2b.elb.fnaldata.org:3128"
1595         export http_proxy=$uswest2baddr
1596         wget http://cvmfs.fnal.gov:8000/cvmfs/cms.cern.ch/.cvmfspublished
1597         returnvalue=$?

```

```

1598     if [ $returnvalue -ne 0 ]
1599     then
1600     echo "Squid Server is not accessible at all" >> $LOG
1601     return 11
1602     else
1603     echo "Squid Server is available in us-west-2b" >> $LOG
1604     addr="http://elb2.us-west-2b.elb.fnaldata.org:3128"
1605     fi
1606
1607     else
1608     echo "Squid Server is not available even in us-west-2b" >> $LOG
1609     return 11
1610     fi
1611
1612 else
1613     echo "Squid Server is available in $zone" >> $LOG
1614     addr="http://elb2.$zone.elb.fnaldata.org:3128" #HK> now, sed command below requires two
... back-slashes, I had restore them here.
1615     fi
1616 # END: new code by HK and ST #####
1617
1618 # make sure /etc/cvmfs/default.local is in place
1619 if [ ! -r /etc/cvmfs/default.local ]
1620 then
1621     echo "/etc/cvmfs/default.local not found" >> $LOG
1622     return 8
1623 fi
1624 # and modify
1625 sed -i -e "s/\\(CVMFS_HTTP_PROXY=\\).*/\\1$addr/" /etc/cvmfs/default.local
1626
1627
1628
1629 # make sure /usr/bin/cvmfs_config is in place
1630 if [ ! -x /usr/bin/cvmfs_config ]
1631 then
1632     echo "/usr/bin/cvmfs_config not executable" >> $LOG
1633     return 9
1634

```

```

1635 fi
1636 # and run it
1637 /usr/bin/cvmfs_config reload >> $LOG 2>&1
1638
1639
1640 # make sure /etc/cvmfs/SITECONF/local/JobConfig/site-local-config.xml is in place
1641 if [ ! -r /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml ]
1642 then
1643     echo "/etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml not found" >>
... $LOG
1645     return 10
1646 fi
1647 # and modify
1648 sed -i -e "s/\\(<proxy url=\\).*/\\1"$addr"/>/"
... /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml
1649
1650
1651 # getting rid of the Fermi-specific crlsquid.fnal.gov and curl-cache.fnal.gov if it's there
1652 if [ -r /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml ]
1653 then
1654     cp -p /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml.fermisav
1655     sed -i -e "s/\\(http_proxy = \\).*/\\1$addr/" /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml
1656     grep -v prepend_url /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml > /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml.temp
1657     cp /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml.temp /etc/cvmfs/SITECONF/T3_US_HEP_Cloud/JobConfig/site-local-config.xml
1658 # start a fetch-curl now at S27 instead of enabling the fetch-curl-boot
1659 # and throw it into background.
1660
1661 # commented out by HK/ST in order to make sure fetch-curl does not populate
... /etc/grid-security/certificates/ with *.r0 files
1663 # nohup /usr/sbin/cvmfs_config < /dev/null >> $LOG 2>&1 &
1664 fi
1665
1666 return 0
1667 }
1668
1669 stop() {

```

```

1670
1671 rm /var/lock/subsys/hepcloud-init-workernode
1672
1673 return 0
1674 }
1675
1676 case "$1" in
1677     start)
1678         start
1679         RETVAL=$?
1680         ;;
1681     stop)
1682         stop
1683         RETVAL=$?
1684         ;;
1685     restart|try-restart|condrestart)
1686         ## Stop the service and regardless of whether it was
1687         ## running or not, start it again.
1688         #
1689         ## Note: try-restart is now part of LSB (as of 1.9).
1690         ## RH has a similar command named condrestart.
1691         start
1692         RETVAL=$?
1693         ;;
1694     reload|force-reload)
1695         # It does not support reload
1696         RETVAL=3
1697         ;;
1698     status)
1699         echo -n $"Checking for service $prog:"
1700         # Return value is slightly different for the status command:
1701         # 0 - service up and running
1702         # 1 - service dead, but /var/run/ pid file exists
1703         # 2 - service dead, but /var/lock/ lock file exists
1704         # 3 - service not running (unused)
1705         # 4 - service status unknown :-(
1706         # 5--199 reserved (5--99 LSB, 100--149 distro, 150--199 appl.)
1707         RETVAL=3

```

```

1708     ;;
1709     *)
1710         echo "Usage: $0
... {start|stop|status|try-restart|condrestart|restart|force-reload|reload}"
1711         RETVAL=3
1712     ;;
1713 esac
1714
1715 exit $RETVAL
1716
1717 -----
1718 -----
1719 -----
1720
1721
1722 File: imagemanagement/step3/hepcloud_imaging_boto.py
1723 import boto3
1724 import sys
1725 import time
1726 import datetime
1727 import os.path
1728
1729 from boto3.session import Session
1730
1731 # HK> we start with the IAM imaging user account, by using aws configure which will update or
... create /root/.aws/configure
1732 # and then, imaging user account will switch to the ManageVMRole role.
1733
1734 def hepcloud_upload( rawfile_pathname ):
1735
1736     if os.path.exists( rawfile_pathname ):
1737         print "%s Exists" % rawfile_pathname
1738     else:
1739         print "%s does not exist" % rawfile_pathname
1740         return -1
1741
1742 # upload the raw image to S3 bucket
1743 rawfilepathname = rawfile_pathname # this might be a full path name

```

```

1744 | head,tail = os.path.split( rawfilepathname )
1745 | # head should be os.path.dirname( rawfilepathname )
1746 | # tail should be os.path.basename( rawfilepathname )
1747 | heps3key = tail # print 'hep3key = ', heps3key
1748 |
1749 |
1750 | #####
1751 | #####
1752 | #####
1753 | # http://boto3.readthedocs.org/en/latest/reference/services/
1754 |
1755 |
1756 | # possible arguments
1757 | #1. 'ManageVMRole'
1758 | #2. 'arn:aws:iam::159076985202:role/ManageVMRole'
1759 | #3. 'vmimagemanagetestbucket'
1760 | #4. HepCloud_Description = ""
1761 | #5. HepCloud_AMI_Name = ""
1762 |
1763 | # get_date = datetime.date.today()
1764 |
1765 | HepCloud_Description = "HepCloud_Imaging_%s" % str( datetime.date.today() )
1766 | HepCloud_Tag          = "HepCloud_%s"          % str( datetime.date.today() )
1767 | HepCloud_AMI_Name     = "HepCloud_AMI_%s"     % str( datetime.date.today() )
1768 |
1769 | # prerequisites
1770 | # arn:aws:iam::159076985202:role/ManageVMRole is created in our RnD account
1771 | #1 check if RoleArn='arn:aws:iam::159076985202:role/ManageVMRole' exists
1772 | session = Session( profile_name = "rnd" )
1773 | client  = session.client('iam')
1774 | list_dir = client.list_roles() # list of dictionaries
1775 |
1776 | rolename_list = [ roledictionary['RoleName'] for roledictionary in list_dir['Roles'] ]
1777 | rolearn_list  = [ roledictionary['Arn']     for roledictionary in list_dir['Roles'] ]
1778 |
1779 | if 'ManageVMRole' in rolename_list:
1780 |     print "the role name %s exists "          % 'ManageVMRole'
1781 | else:

```

```

1782 |     print "the role name %s does not exist " % 'ManageVMRole'
1783 |
1784 | if 'arn:aws:iam::159076985202:role/ManageVMRole' in rolearn_list:
1785 |     print "the arn %s exists "              % 'arn:aws:iam::159076985202:role/ManageVMRole'
1786 | else:
1787 |     print "the arn %s does not exist" % 'arn:aws:iam::159076985202:role/ManageVMRole'
1788 |
1789 |
1790 | # S3 Resource
1791 | heps3bucket = 'vmimagemanagetestbucket' # this is fixed
1792 | # session = Session(profile_name="hwk")
1793 | client = session.client('s3')
1794 | list_dir = client.list_buckets() #
1795 | ... http://boto3.readthedocs.org/en/latest/reference/services/s3.html#S3.Client.list_buckets
1796 | bucket_list = list_dir['Buckets']
1797 | namelist = [ x['Name'] for x in bucket_list ]
1798 | if heps3bucket in namelist:
1799 |     print "our bucket %s exists"          % heps3bucket
1800 | else:
1801 |     print "our bucket %s does NOT exist" % heps3bucket
1802 |
1803 | #####
1804 | #####
1805 | #####
1806 |
1807 | # using boto3 default session
1808 | client = boto3.client('sts')
1809 | print 'assume_role call'
1810 | # role switching
1811 | response = client.assume_role( RoleArn='arn:aws:iam::159076985202:role/ManageVMRole',
1812 | ... RoleSessionName='sessiontest' )
1813 | # have to check if role-switching was successful
1814 | role_AK_id = response['Credentials']['AccessKeyId']
1815 | role_AK_sc = response['Credentials']['SecretAccessKey']
1816 | role_AK_tk = response['Credentials']['SessionToken']
1817 |

```

```

1818
1819
1820
1821 #####
1822 #####
1823 #####
1824
1825 # New Session in West 2
1826     print 'Opening Session with temporary key'
1827     session_w2 = Session(aws_access_key_id=role_AK_id, aws_secret_access_key=role_AK_sc,
...
1828     aws_session_token=role_AK_tk, region_name='us-west-2') # have to check of the session-creation
1829     was successful
1830
1831 # back to ManageVMRole role
1832     heps3 = session_w2.resource('s3') # session.client('s3') works but client.Object does not
...
1833     exist, only resource.Object..
1834
1835     hep3object = heps3.Object(heps3bucket , heps3key) #heps3.Object(heps3bucket ,
...
1836     hep3key).put( Body=open( rawfilepathname, 'rb') )
1837     hep3object.upload_file( rawfilepathname ) # hep3object.put( Body=open( rawfilepathname,
...
1838     'rb') )
1839     print 'put submitted, now calling wait_until_exists'
1840     hep3object.wait_until_exists() # have to check if s3 put was initiated successfully
1841     print 'return from wait_until_exists'
1842
1843
1844
1845
1846 # EC2 Client from the same West 2 session for Import Image
1847     print "Now importing from S3 to AMI in West 2"
1848     ec2_w2 = session_w2.client('ec2')
1849
1850     response = ec2_w2.import_image(

```

```

1851         Description = HepCloud_Description,
1852         DiskContainers = [
1853             {
1854                 'Description': HepCloud_Description,
1855                 'UserBucket': {
1856                     'S3Bucket': heps3bucket,
1857                     'S3Key': heps3key
1858                 },
1859             }
1860         ] # end of import_image
1861
1862     ImportTaskId = response['ImportTaskId'] # print 'import task id = ', ImportTaskId
1863
1864
1865 # now polling the import_image task
1866     print "import task polling"
1867     response = ec2_w2.describe_import_image_tasks( ImportTaskIds=[ ImportTaskId ] )
1868     status=response['ImportImageTasks'][0]['Status']
1869
1870     while status != 'completed':
1871         time.sleep(60)
1872         response=ec2_w2.describe_import_image_tasks( ImportTaskIds=[ ImportTaskId ] )
1873         status=response['ImportImageTasks'][0]['Status']
1874
1875     print "finally"
1876     print status
1877 # polling is over
1878
1879
1880
1881 # Now, the AMI is available
1882     response=ec2_w2.describe_import_image_tasks( ImportTaskIds=[ ImportTaskId ] )
1883     AMI_W2 = response['ImportImageTasks'][0]['ImageId']
1884
1885
1886 # EC2 Resource from the same session for
1887     print "Tagging West 2 AMI"
1888     ec2_w2_resource = session_w2.resource('ec2')

```

```

1889     hep_west2_image = ec2_w2_resource.Image( AMI_W2 )
1890     tag = hep_west2_image.create_tags( Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1891
1892 # account sharing
1893     print "Sharing West2 image with all other 3 accounts"
1894     response = hep_west2_image.modify_attribute( Attribute='tags', OperationType='add',
LaunchPermission={ 'Add': [ { 'UserId': '486926498429' }, { 'UserId': '950490332792' }, {
...
'UserId': '229161804233' }, ] } )
1895 # in order to tag these shared AMI that are shared by other 3 accounts(CMS, NOVA, Fermilab)
1896 # I need to open a new custom session with a profile
1897 # /root/.aws/credentials must contain the following profiles
1898     session_tmp_cms = Session(profile_name="cms")
1899     session_tmp_nov = Session(profile_name="nova")
1900     session_tmp_fna = Session(profile_name="fnal")
1901     session_tmp_cms.resource('ec2', region_name='us-west-2').Image( AMI_W2 ).create_tags(
...
Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1902     session_tmp_nov.resource('ec2', region_name='us-west-2').Image( AMI_W2 ).create_tags(
...
Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1903     session_tmp_fna.resource('ec2', region_name='us-west-2').Image( AMI_W2 ).create_tags(
...
Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1904
1905
1906
1907 #####
1908 #####
1909 ## A New Session for West 1
1910 # now copying
1911     session_w1 = Session(aws_access_key_id=role_AK_id, aws_secret_access_key=role_AK_sc,
...
aws_session_token=role_AK_tk, region_name='us-west-1')
1912     print "Now copying from W2 to W1"
1913
1914 # EC2 Client from West1 Session
1915     ec2_w1_client = session_w1.client('ec2')
1916     response_w1 = ec2_w1_client.copy_image( SourceRegion='us-west-2',
...
SourceImageId=AMI_W2, Name=HepCloud_AMI_Name, Description=HepCloud_Description
...
)
1917     print "W1 copy wait starts"
1918     waiter_w1 = ec2_w1_client.get_waiter('image_available')

```

```

1919     AMI_W1 = response_w1['ImageId']
1920     waiter_w1.wait( ImageIds = [ AMI_W1 ] )
1921     print "W1 copy wait ends"
1922
1923
1924 # EC2 Resource from West1 Session
1925     print "W1 tagging"
1926     ec2_w1_resource = session_w1.resource('ec2')
1927     hep_west1_image = ec2_w1_resource.Image( AMI_W1 )
1928     tag = hep_west1_image.create_tags( Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag },
...
] )
1929     print "W1 tagging successful"
1930
1931 # account sharing
1932     print "Sharing West1 image with all other 3 accounts"
1933     response = hep_west1_image.modify_attribute( LaunchPermission={ Attribute='tags',
...
OperationType='add', 'Add': [ { 'UserId': '486926498429' }, { 'UserId': '950490332792' }, {
'UserId': '229161804233' }, ] } )
1934     session_tmp_cms.resource('ec2', region_name='us-west-1').Image( AMI_W1 ).create_tags(
...
Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1935     session_tmp_nov.resource('ec2', region_name='us-west-1').Image( AMI_W1 ).create_tags(
...
Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1936     session_tmp_fna.resource('ec2', region_name='us-west-1').Image( AMI_W1 ).create_tags(
...
Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1937
1938
1939 #####
1940 #####
1941 #####
1942 ## A New Session for East 1
1943     print "Now copying from W2 to E1"
1944     session_e1 = Session(aws_access_key_id=role_AK_id, aws_secret_access_key=role_AK_sc,
...
aws_session_token=role_AK_tk, region_name='us-east-1')
1945
1946 # EC2 Client from East 1 Session
1947     ec2_e1_client = session_e1.client('ec2')
1948     response_e1 = ec2_e1_client.copy_image( SourceRegion='us-west-2',
...
SourceImageId=AMI_W2, Name=HepCloud_AMI_Name, Description=HepCloud_Description )

```

```

1949     print "E1 copy wait starts"
1950     waiter_e1 = ec2_e1_client.get_waiter('image_available')
1951     AMI_E1 = response_e1['ImageId']
1952     waiter_e1.wait( ImageIds=[ AMI_E1 ] )
1953     print "E1 copy wait ends"
1954
1955 # EC2 Resource from East 1 Session
1956     print "E1 tagging"
1957     ec2_e1_resource = session_e1.resource('ec2')
1958     hep_east1_image = ec2_e1_resource.Image( AMI_E1 )
1959     tag = hep_east1_image.create_tags(      Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag },
... ] )
1960     print "E1 tagging ends"
1961
1962
1963 # account sharing
1964     print "Sharing East1 image with all other 3 accounts"
1965     response = hep_east1_image.modify_attribute( LaunchPermission={ Attribute='tags',
... OperationType='add', 'Add': [ { 'UserId': '486926498429' }, { 'UserId': '950490332792' }, {
... 'UserId': '229161804233' }, ] } )
1966     session_tmp_cms.resource('ec2', region_name='us-east-1').Image( AMI_E1 ).create_tags(
... Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1967     session_tmp_nov.resource('ec2', region_name='us-east-1').Image( AMI_E1 ).create_tags(
... Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1968     session_tmp_fna.resource('ec2', region_name='us-east-1').Image( AMI_E1 ).create_tags(
... Tags=[ { 'Key': 'Name', 'Value': HepCloud_Tag }, ] )
1969
1970
1971
1972
1973
1974 #####
1975 #####
1976 #####
1977 def hepcloud_main(argv=None):
1978     hepcloud_upload(argv[1])
1979 if __name__ == "__main__":
1980     argv = sys.argv

```

```

1981     hepcloud_main(argv)
1982
1983 -----
1984 -----
1985 -----
1986
1987
1988 File: imagemanagement/step3/README
1989 mkdir /opt/gcso/awsexport/
1990 mkdir /data/
1991
1992
1993 -----
1994 -----
1995 -----
1996
1997
1998 File: imagemanagement/step3/run-boto.sh
1999 #!/bin/bash
2000
2001 # description of positional arguments
2002 # 1. /opt/gcso/awsexport : the directory where the codes will find everything
2003 # 2. oneadmin@fclheadgpvm01:/var/lib/one/datastores/102/f42715e37dbbe858af596dfe8827be02 : the
... source qcow2 image in FermiCloud Image Repository
2004 # 3. $newkernelversion : I need to check again, with boto we might not need this parameter any
... longer. It used to be required by the old ec2iin command
2005 # 4. newtest : name of the image files qcow2 and raw
2006 # 5. ManageVMRole-Test : a string for AWS description
2007 # 6. /var/cache/cvmfs2 : when you are using certain instance types of AWS(c3.large), the
... second ephemeral store will be available as /dev/xvdc, and we are using it for CVMFS cache
2008 # - mkdir -p /var/cache/cvmfs2 in the image
2009 # - echo 'CVMFS_CACHE_BASE='$2'' >>/etc/cvmfs/default.local
2010 # - echo '/dev/xvdc          '$2'                ext3          defaults          0 0' >> fstab
2011
2012 kinit    -k -t /var/adm/krb5/cloudadminpp.keytab
... cloudadmin/cron/fermicloudpp.fnal.gov@FNAL.GOV
2013 newkernelversion=`ssh -l oneadmin fclheadgpvm01.fnal.gov cat /tmp/kernelversion.txt`
2014 echo $newkernelversion

```

```

2015 /opt/gcso/awsexport/Convert-boto.py /opt/gcso/awsexport
... oneadmin@fc1headgpvm01:/var/lib/one/datastores/102/f42715e37dbbe858af596dfe8827be02
... $newkernelversion newestest ManageVMRole-Test /var/cache/cvmfs2
2016
2017 -----
2018 -----
2019 -----
2020
2021
2022 File: imagemanagement/step3/site-local-config.xml
2023 <site-local-config>
2024 <site name="T3_US_HEP_Cloud">
2025   <event-data>
2026     <catalog
... url="trivialcatalog_file:/cvmfs/cms.cern.ch/SITECONF/local/PhEDEx/storage.xml?protocol=xrd"/>
2027     <catalog
... url="trivialcatalog_file:/cvmfs/cms.cern.ch/SITECONF/local/PhEDEx/storage.xml?protocol=
... fallbackxrd"/>
2028   </event-data>
2029   <source-config>
2030     <!--statistics-destination name="cms-udpmon-collector.cern.ch:9331" /-->
2031   </source-config>
2032   <local-stage-out>
2033     <se-name value="cmssrmdisk.fnal.gov"/>
2034     <command value="stageout-xrdcp-fnal"/>
2035     <catalog
... url="trivialcatalog_file:/cvmfs/cms.cern.ch/SITECONF/local/PhEDEx/storage.xml?protocol=
... writexrd"/>
2036     <phedex-node value="T3_US_HEP_Cloud"/>
2037   </local-stage-out>
2038   <calib-data>
2039     <frontier-connect>
2040       <load balance="proxies"/>
2041       <proxy url="http://AMAZON.URL.GOES.HERE:3128"/>
2042       <backupproxy url="http://cmsbproxy.fnal.gov:3128"/>
2043       <backupproxy url="http://cmsbproxy01.fnal.gov:3128"/>
2044       <backupproxy url="http://cmsbproxy02.fnal.gov:3128"/>
2045       <server url="http://cmsfrontier.cern.ch:8000/FrontierInt"/>

```

```

2046       <server url="http://cmsfrontier.cern.ch:8000/FrontierInt"/>
2047       <server url="http://cmsfrontier1.cern.ch:8000/FrontierInt"/>
2048       <server url="http://cmsfrontier2.cern.ch:8000/FrontierInt"/>
2049       <server url="http://cmsfrontier3.cern.ch:8000/FrontierInt"/>
2050       <server url="http://cmsfrontier4.cern.ch:8000/FrontierInt"/>
2051     </frontier-connect>
2052   </calib-data>
2053 </site>
2054 </site-local-config>
2055
2056 -----
2057 -----
2058 -----
2059
2060
2061 File: imagemanagement/step3/storage.xml
2062 <storage-mapping>
2063
2064 <!-- PRODUCTION BEGIN -->
2065
2066 <!-- Xrootd -->
2067 <lfn-to-pfn protocol="xrd"
2068   destination-match=".*" path-match="^/+store/unmerged/(.*)"
... result="root://cmseos.fnal.gov//lustre/unmerged/$1"/>
2069
2070 <lfn-to-pfn protocol="xrd"
2071   destination-match=".*" path-match="/+lustre/(.*)"
... result="root://cmseos.fnal.gov//lustre/$1"/>
2072
2073 <lfn-to-pfn protocol="fallbackxrd"
2074   destination-match=".*" path-match="/+lustre/(.*)"
... result="root://cmseos.fnal.gov//lustre/$1"/>
2075
2076 <lfn-to-pfn protocol="writexrd"
2077   destination-match=".*" path-match="/+lustre/(.*)"
... result="root://cmseos.fnal.gov//lustre/$1"/>
2078 <lfn-to-pfn protocol="writexrd"
2079   destination-match=".*" path-match="/+store/temp/user/(.*)"

```

```
2079... result="root://cmseos.fnal.gov/eos/uscms/store/temp/user/$1"/>
2080 <lfn-to-pfn protocol="writexrd"
2081   destination-match="*" path-match="/+store/unmerged/(.*)"
...   result="root://cmseos.fnal.gov/lustre/unmerged/$1"/>
2082
2083 <lfn-to-pfn protocol="writexrd"
2084   path-match="^/+lustre/unmerged/logs/prod/(.*)"
...   result="root://cmseos.fnal.gov/lustre/unmerged/logs/prod/$1"/>
2085
2086 </storage-mapping>
2087
2088
2089 -----
2090 -----
2091 -----
2092
2093
2094
```