

# 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

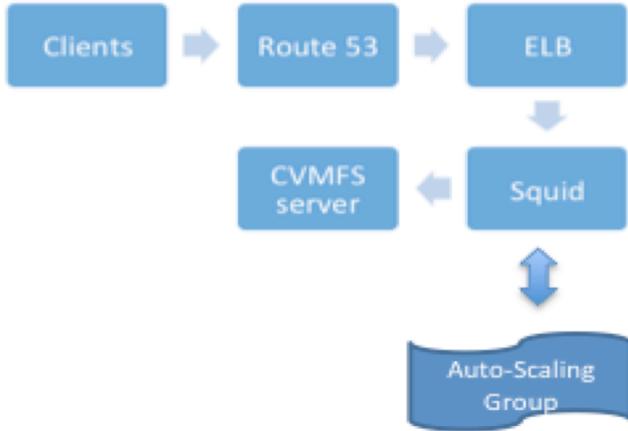


Figure 1: CVMFS Client/Server Architecture on AWS

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.

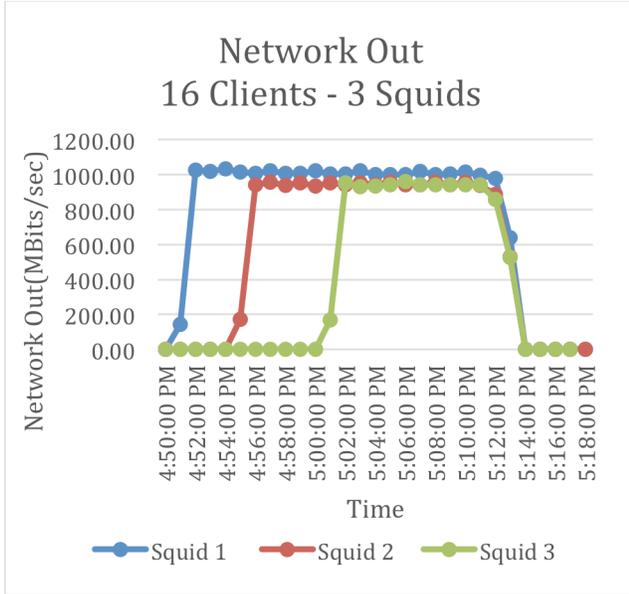


Figure 2: Network throughput of Squid server

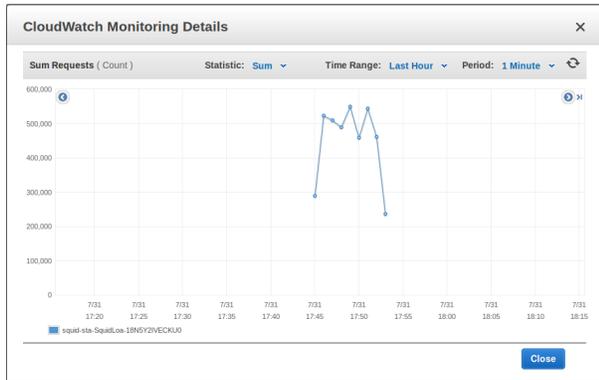


Figure 3: Load Balancer Requests per Minute

## 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

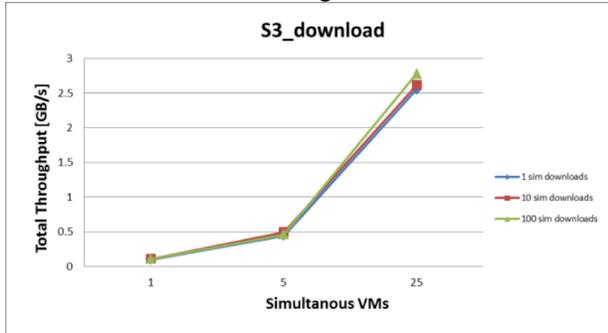
The bandwidth tests have been carried out through the usage of custom made scripts that employ the same transfer protocols and storage systems that will be adopted during the execution of a CMS job.

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.

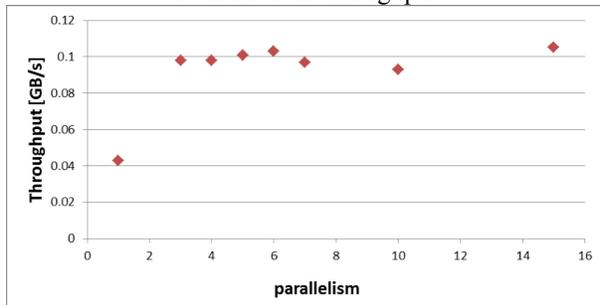
In order to store the final results of the CMS job, FermiGrid storages have been considered. The `globus-url-copy` and `xrdcp` commands were adopted respectively to transfer to 2 different servers. Due to the high latency from Amazon to Fermilab, the file transfers had to be carried out by using

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.

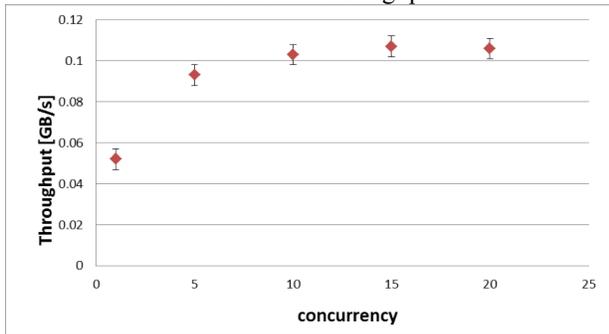
**Figure 4:** Download bandwidth throughput test from Amazon S3 to c3.2xlarge instances



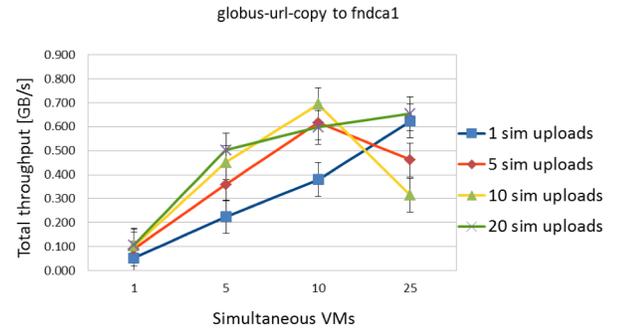
**Figure 5:** Study of the effect of the parallelism parameter over the total throughput



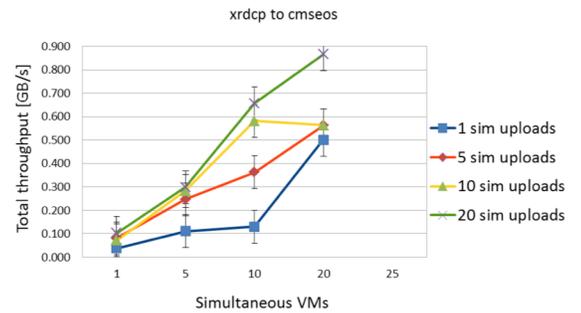
**Figure 6:** Study of the effect of the concurrency parameter over the total throughput



**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 XEON 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 XEON X5355	2.66	0.0173	0.0173	13.2	13.2
prvm0190	4	Intel XEON 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.