Managed by Fermi Research Alliance, LLC for the U.S. Department of Energy Office of Science

Benchmarking of public and local cloud resources

Computing division

2015 Italian Summer internship

Davide Grassano
Università degli studi di Roma Torvergata
Materials Science and Technology

UNIVERSITA' degli STUDI di ROMA
TOR VERGATA

Supervisor
Gabriele Garzoglio

Index

1. Introduction

This report deals with my work at the Computing Division carried out during the Fermilab summer internship 2015 period. In particular, my aim was to contribute to the expansion of the current capabilities of the Cloud Project.

As of the day of my arrival at Fermilab, the FermiCloud infrastructures and its management services were already running, and the project was undergoing through the phase of establishing production services and amplifying the system capabilities in order to match users need and request. One of the main focuses during my internship period has been to take part in the realization of a cloud service capable or running full length CMS jobs. While pursuing this objective, a part of the necessary studies was the benchmarking of both local and commercial cloud resources which is the job that I carried out.

This involved analyzing the proficiency of various machine at running pieces of CMS jobs, and the study of the bandwidth throughput from and toward various available storages, in order to decide the best way to handle the data needed and produced from the jobs.

During my work, I wrote various scripts with the aim of automatizing the process of launching virtual machine on the cloud, executing the benchmarks and cropping the resulting data.

This was a necessary step, especially regarding the bandwidth benchmarks that needed to be synchronized in order to accurately reproduce what might be the amount of data transfer required for a CMS job.

2. Cloud computing

Before introducing the concept of cloud computing, it is necessary to expand on one of its predecessor: grid computing, known also as distributed computing.

2.1. Grid Computing

Grid computing is the collection of computer resources in a unified infrastructure where all the machines will work in order to achieve a common goal. This system requires for the usage of complete computers, connected by a conventional network interface, allowing for the harnessing of unused resources.

A grid, such as FermiGrid, will present multiple worker nodes, each specialized in a particular task. This represents a major difference when compared to cluster computing, where each node is set to perform the same task.

One of the disadvantages of a grid is the fact that the nodes may not have high-speed connection between each other, due to the grid's intrinsic nature of having a heterogeneous geographical distribution of resources.

Cloud computing improves on Grid computing, by adding an on-demand resource provisioning and a layer of virtualization, that can operate on different scales, from software, to OS. The users can access the cloud through the use of a client, known as thin client, which depends heavily on a server set on a different computer.

This results in advantages, for both the host and the users of a cloud system. The first will be able to maximize the efficiency of his facility and the achievement of an economy of scale, while the users will be able to pay only for the services they need on an on-demand basis. This represents a versatile system that companies can use to expand their computing power only when needed with no upfront commitment of money.

2.2. Models of cloud services

2.2.1. Software as a Service

Also known as on-demand-software, the SaaS model provides on a pay-per-use basis the access to software and databases to its users.
On the host ends there will be a load balancer, tasked with the distribution of the workload across available VMs, while the user will see a single access point to the provided service, greatly simplifying the running of a task on its end.
This model constitutes one of the biggest shares of the cloud market, since it enables its users to access software from a third party, through interfaces as simple as a web browser.
Examples: Google Apps, Salesforce, Workday.

2.2.2. Platform as a Service

In the PaaS model, the cloud provider delivers a computing platform to his users, which generally includes an OS and all the required tools for programming. This allows for a reduction in complexity that software developer will have to deal with in higher level

programming, while also outsourcing the costs and manpower required for the maintenance of the hardware.
Examples: Microsoft Azure, Google App Engine

2.2.3. Infrastructure as a Service

The IaaS model provides its user with access to virtualized or bare metal machine within the cloud, allowing for the acquisition of extra computing power without having to buy new hardware. This is especially in systems, where the computing power required over time will present spikes.
Examples: FermiCloud, Amazon Web Services (AWS), Google Compute Engine

2.3. The Cloud Project

One of the already achieved aims of the FermiCloud project was to establish a scientific private IaaS cloud at Fermilab, in order to add to the already present functionalities of the FermiGrid. This allows for the virtualization of worker nodes, and sending Virtual Machines as jobs, within the cloud, while also permitting an on-demand allocation of resources, without the need of the intervention of a system administrator. This model is more dynamical when compared with other virtualization utility used on the Grid, in the fact that VMs comes into being at the moment of their request, and cease to exist when the job is done, freeing the resources back into the cloud.
Another of the aims of the project is to realize a facility that can be interoperable with other large cloud-based user and facilities, making it possible for the sharing of resources between scientific institutions while they are not being employed by the main host.
The FermiCloud can be further improved by switching from a private cloud to a hybrid one through the combination with a commercially available cloud like AWS. This would allow for an on-demand acquisition of resources when confronted with workloads that exceed the current capabilities of the system, a technique known as cloud bursting.

3. Benchmarking

Benchmarking consists in running a series of programs and application on a machine, either bare metal or virtual, in order to assess its capabilities, with the aim of establishing a metric that can be used to compare different hardware.

This can be achieved by running the same sets of programs on different machine and compare either the time it took for the whole benchmark to complete, or the percentage of job completion (or other related parameters) the machines were able to reach in a fixed amount of time.

In order to compare hardware on a global scale, generic benchmarks are used. This are meant to run on a wide variety of architectures, and to test a wide range of features that can be the CPU capabilities under stress, the disk or RAM input/output per seconds (iops), the capabilities of the compiler, the bandwidth and so on. This list represents a set of features that allows the comparison of resources on a generic scale.

Generic benchmarks are important when buying new machine that most likely won't be running a single kind of job, but a wide variety, since having high generic specs will means that the system is more versatile.

Unfortunately, generic benchmarks do not always represent accurately the kind of workload that the user wishes to run and for this reason specific benchmarks can also be used. Their purpose is to emulate the whole workload imposed by the job or just part of it. This gives results that are more suited for determining which system will be the best for that particular application. Doing so is especially important when dealing with an on-demand service, since machines are allocated only for the duration of the job where the most suited one can be chosen on a case by case basis.

Having benchmark data is also fundamental when compiling a business plan in order to get funding, where the developer has to guarantee to be able to provide at minimum the performances required by the stakeholder.

During my work, both generic and specific benchmarks have been used, in order to establish a metric to compare machines on a global scale, while also determining which one of them would be the best at running a full length CMS job, by taking into account the efficiency over price.

Here are the benchmarks that have been used, some of them being standards, while others being custom made:

3.1. $t\bar{t}$bar_gensim

The gensim benchmark simulates one of the initial phases of a CMS job and consists in the generation of 150 ttbar events. The test can generate up to 100GB of files that are meant to be used in other steps of the jobs, such has the reconstruction phase.

The results of the benchmark are given in ttbar/s and, the higher the value, the better the machine will be at running a CMS job.

3.2. Hepspec06

SPEC (Standard Performance Evaluation Corporation) produced a portable collection of benchmark with the focus on compute intensive performances, meaning that the CPU, memory architecture and compilers of the system will be put under test.
The hepspec06 is a subset of the SPEC benchmarks collection defined by the all_cpp command. Its purpose is to analyze the integer and floating point performances of the c++ compiler of the system. It was chosen because the components tested by this benchmark are similar to those used in a CMS job, which should be reflect on the comparison of hepspec and gensim results. These collections are widely used across the globe, presenting a great repository of results from a great variety of hardware, allowing their comparison through the SPEC parameter, which is obtained by calculating the ratio of a standard value divided by the running time for each benchmark of the package and then calculating the geometric mean of all these values.

3.3. Bandwidth throughput tests

During the execution of a CMS job, data will need to be stored and read to and from various sources of storage. The speed at which these operations are carried out can greatly impact the efficiency, and therefore the cost, of the job. For this matter, the Cloud Project has to guarantee a minimum of bandwidth in order to satisfy the needs of a full CMS job. With this in mind, various custom benchmarks had to be written, in order to assess the bandwidth up and download throughput with respect to the following storage systems: Amazon S3, FermiGrid, cmseos.
The first one is of great interest for storing the pileup file produced from the results of the gensim phase and passing them to the reconstruction one. The AWS Command Line Interface(CLI) was used for the transfer of the files from a local disk to an S3 bucket.
The other storage systems represent 2 of the potential grid system to use in order to store the data produced by a job. FermiGrid use a gridftp protocol, and its nodes could be accessed through the use of the globus-url-copy command, while cmseos required either the XRootD or SRM (Storage Resource Management) libraries to be accessed.
In both cases the request were being sent to a server hosting a dCache that would distribute the requests through various nodes.
Within both libraries was the option to transfer files using parallel streams. A case study of the effect of the number of parallel stream and other parameters can be found in chapter 5.
The results are reported as total throughput per number of simultaneous uploads, in function of the number VMs that were running the benchmark simultaneously.

4. Automation of the process

Having to run multiple benchmarks on a wide variety of machines can be easily prone to the occurrence of human errors if carried out manually. This prompted me to automatize the benchmarking process through the writing of various .sh scripts. This also allowed for the execution of a greater number of benchmark that it would’ve been if all the operation were to be carried out manually. The automation of the process was also mandatory in the later phase of the work where, in order to accurately simulate the data transfer of a CMS job, all of the VMs executing the bandwidth benchmark had to be synchronized with each other.

The main script produced is the aws_launch_benchmark.sh capable of launching the requested AWS instances, initialize them and transfer and run a run-benchmark_name.sh script capable of setting up all the needed files and running the benchmark on the VM sequentially, or synchronized if needed. A series of checks and a README file have been introduced in order to make the script as user friendly as possible, so that it could be easily used by other users as well.

One of the other scripts is the crop_results.sh, capable of collecting the results data from all the VMs that were instantiated during the last run of aws_launch_benchmark.sh, while also doing some preliminary analysis on it, so to present the data in a more ready to use format.

5. Obtained results

This chapter is focused on presenting the obtained results and discussing their analysis.

5.1. $t\bar{t}$bar_gensim

The gensim benchmark was run on a wide variety of AWS and local machine testing for both single core and all cores performances. Part of the results can be observed in Table 1.

	run	ttbar/s per core	total ttbar/s	Eff. Loss	N° CPU	RAM(GB)
m3.xlarge	All cores	0,0139	0,0557	0,475	4	15
	1 core	0,0265	0,0265			
m3.2xlarge	All cores	0,0139	0,111	0,470	8	30
	1 core	0,0261	0,0261			
m4.xlarge	All cores	0,0201	0,0806	0,431	4	16
	1 core	0,0354	0,0354			
m4.2xlarge	All cores	0,0191	0,153	0,459	8	32
	1 core	0,0354	0,0354			
m4.4xlarge	All cores	0,0198	0,317	0,438	16	64
	1 core	0,0353	0,0353			
c3.xlarge	All cores	0,0153	0,0611	0,478	4	7,5
	1 core	0,0293	0,0293			
c3.2xlarge	All cores	0,0153	0,122	0,475	8	15
	1 core	0,0291	0,0291			
c3.4xlarge	All cores	0,0149	0,239	0,494	16	30
	1 core	0,0295	0,0295			
c4.xlarge	All cores	0,0228	0,091	0,446	4	7,5
	1 core	0,0412	0,0412			
c4.2xlarge	All cores	0,0226	0,181	0,427	8	15
	1 core	0,0395	0,0395			
c4.4xlarge	All cores	0,0205	0,327	0,491	16	30
	1 core	0,0402	0,0402			
r3.xlarge	All cores	0,0151	0,060	0,448	4	30,5
	1 core	0,0273	0,0273			
r3.2xlarge	All cores	0,0150	0,120	0,443	8	61
	1 core	0,0269	0,0269			
r3.4xlarge	All cores	0,0146	0,233	0,463	16	122
	1 core	0,0271	0,0271			
cc2.8xlarge	All cores	0,0141	0,450	0,478	32	60,5
	1 core	0,0269	0,0269			

Table 1: Collecion of results obtained with the gensim benchmark from AWS instances

One of the problems of the benchmark that arose from the data analysis is the amount of efficiency loss when running the All cores test being close to 45%. The reason for such a high value is the fact that this benchmark, when testing more than one core, is not using not launching a multithreaded process, but many single-threaded ones. This wasn't a matter of concern, because an actual CMS job should make proper use of multithreading.

At the end, the gensim results were confronted with AWS on-demand pricing in order to obtain an efficiency/cost parameter.

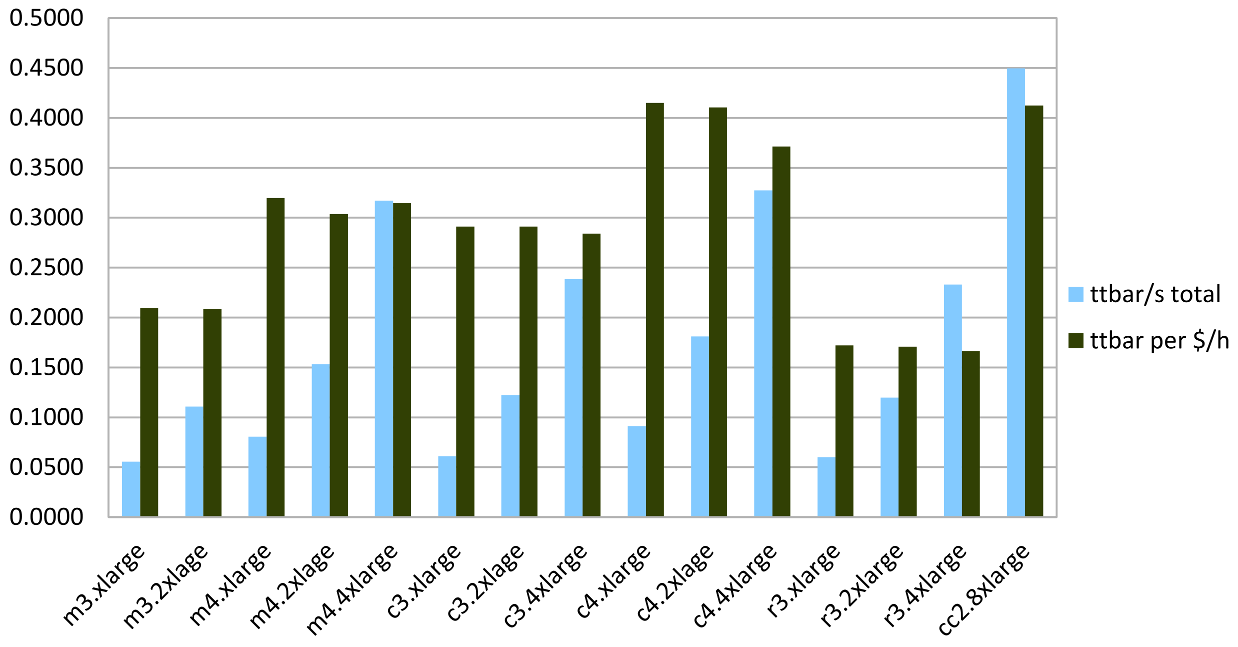

Figure 1: Charts of the total ttbar and ttbar/cost values for every AWS instance analyzed

The values reported in Figure 1 are not the only ones that will be considered in choosing the desired instance, but will need to weighed together with many other parameters mostly dependent on considerations over the spot market.

5.2. hepspec06

As mentioned in chapter 3, the results for the hepspec06 benchmark should be similar to those reported for the gensim one. This can be seen if comparing the results presented in the past paragraph with those in Table 1 and Figure 2.

	run	hepspec per core	total hepspec	Eff. Loss	N° CPU	RAM(GB)
m3.xlarge	All cores	14,29	57,15	0,416	4	15
	1 core	24,48	24,48			
m3.2xlarge	All cores	12,20	97,64	0,490	8	30
	1 core	23,93	23,93			
m4.xlarge	All cores	16,13	64,53	0,426	4	16
	1 core	28,13	28,13			
m4.2xlarge	All cores	15,14	121,13	0,450	8	32
	1 core	27,53	27,53			
m4.4xlarge	All cores	13,53	216,56	0,513	16	64
	1 core	27,80	27,80			
c3.xlarge	All cores	14,86	59,42	0,446	4	7,5
	1 core	26,79	26,79			
c3.2xlarge	All cores	14,73	117,82	0,466	8	15
	1 core	27,58	27,58			
c3.4xlarge	All cores	13,23	211,65	0,527	16	30
	1 core	27,97	27,97			

	run	hepspec per core	total hepspec	Eff. Loss	N° CPU	RAM(GB)
c4.xlarge	All cores	17,46	69,86	0,392	4	7,5
	1 core	28,74	28,74			
c4.2xlarge	All cores	16,55	132,39	0,469	8	15
	1 core	31,17	31,17			
c4.4xlarge	All cores	14,80	236,81	0,527	16	30
	1 core	31,32	31,32			
r3.xlarge	All cores	15,51	62,03	0,415	4	30,5
	1 core	26,51	26,51			
r3.2xlarge	All cores	14,22	113,72	0,456	8	61
	1 core	26,15	26,15			
r3.4xlarge	All cores	12,68	202,87	0,522	16	122
	1 core	26,51	26,51			
cc2.8xlarge	All cores	11,21	358,84	0,540	32	60,5
	1 core	24,36	24,36			

Table 2: Collecion of results obtained with the hepspec06 benchmark from AWS instances

This benchmarks were run in a fashion similar to that of the gensim one, where All cores run were executed by launching many single-threaded processes. As expected, the efficiency loss values are comparable to those of the gensim benchmark.

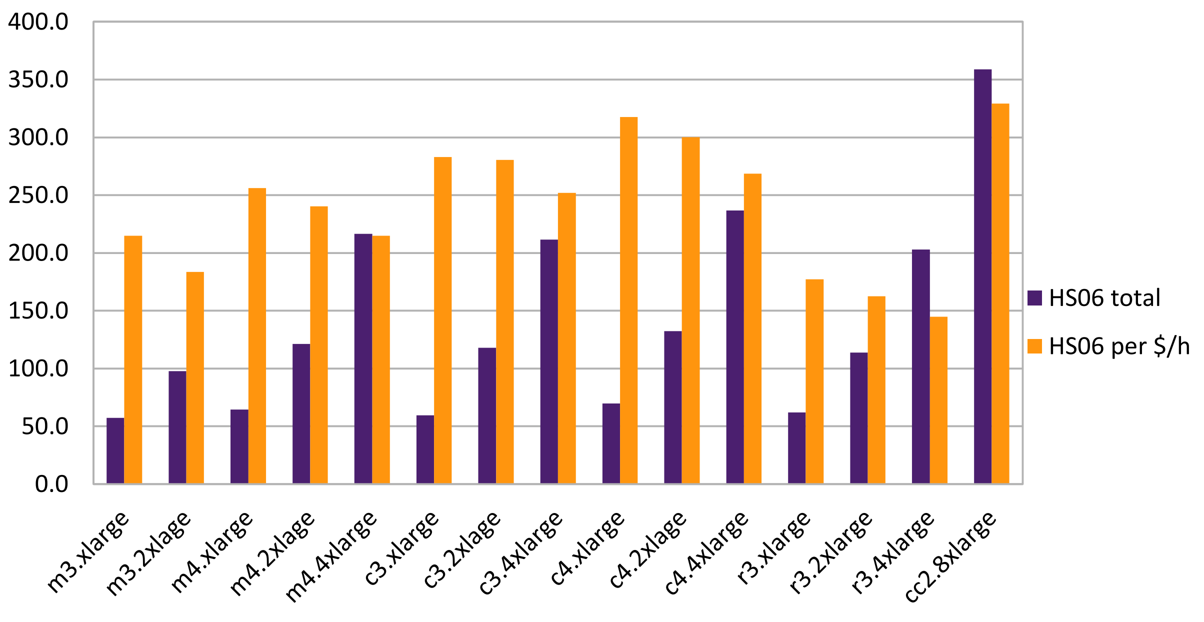

Figure 2: Charts of the total HS06 and HS06/cost values for every AWS instance analyzed

As it can be observed from Figure 2, even if the scale is different, the shape of the graph closely replicates that of the gensim benchmark, result that was expected from how the hepspec06 was chosen.

5.3. Bandwidth throughput tests

The first thing to be tested was the bandwidth throughput from Amazon S3 storage, to c3.2xlarge instances.

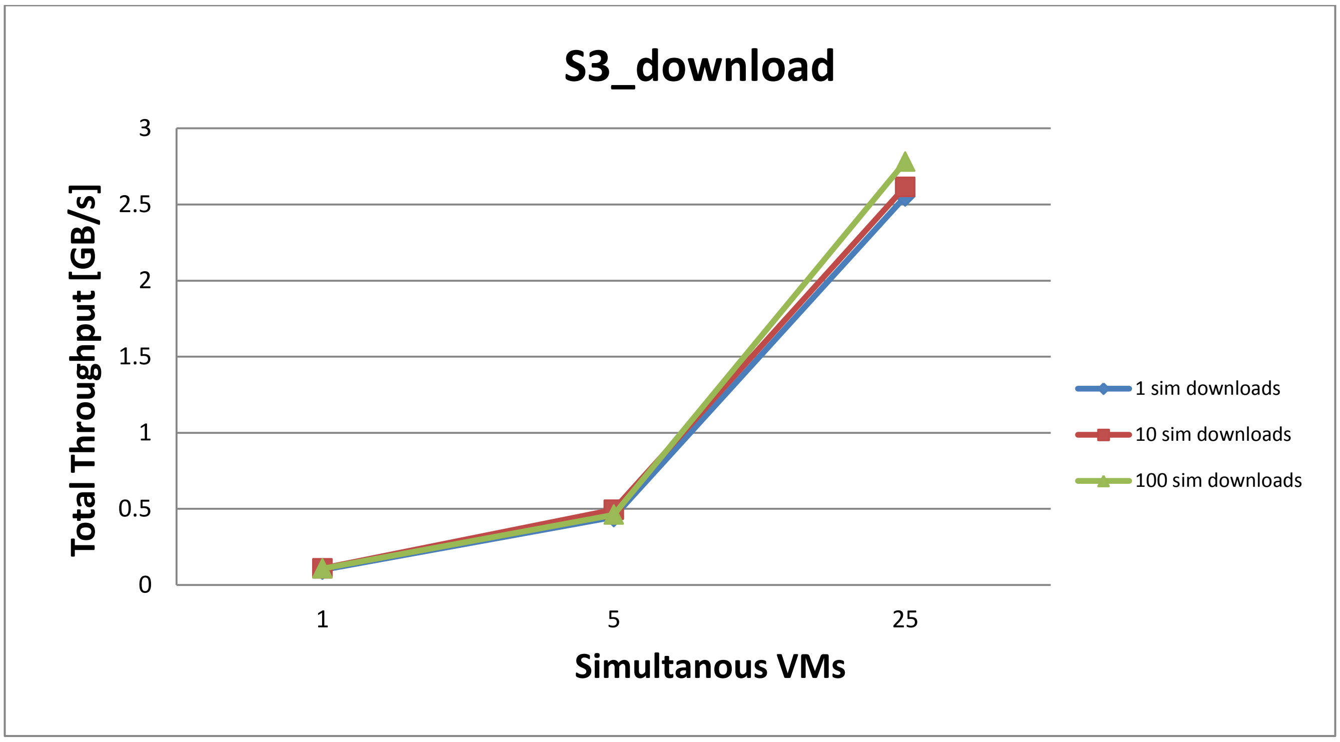

Figure 3: Download bandwidth throughput test from Amazon S3 to c3.2xlarge instances

What was observed from the results shown in Figure 3 is the fact that the only limit imposed on the total throughput, was the maximum bandwidth of 1Gbit/s per c3.2xlarge instance. No degradation of this value was observed with the increase in number of simultaneous VMs and uploads.

The download data was the one of greater interest, since the intent is to use S3 to read pileup files during the reconstruction phase.

Successively the upload bandwidth throughput toward FermiGrid was tested, in order to determine the best way to store the final results.

Before doing so, a study of the effect of the parameters of the globus-url-copy and xrdcp commands had to be done, in order to determine the best number of parallel streams to use when uploading files.

To achieve this, the total throughput in function of the parallelism was analyzed, while simultaneously uploading 20 files from 1 VM.

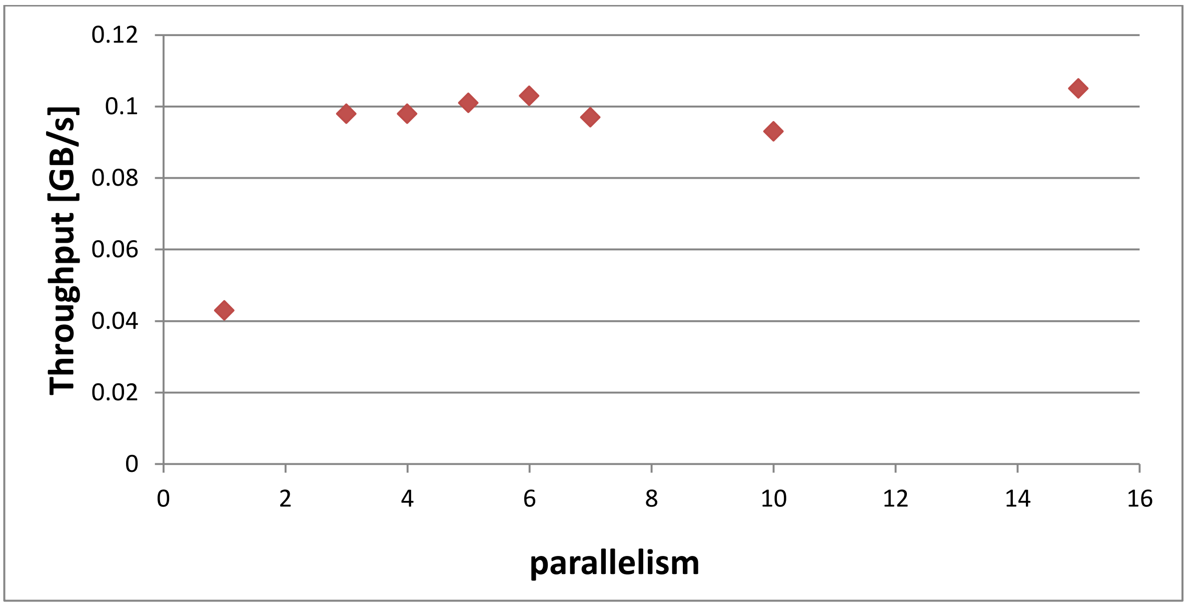

Figure 4: Study of the effect of the parallelism parameter over the total throughput

The results of this study (Figure 4) show that for values of parallelism over 3-4, the total throughput tends to stabilize, making it not worth to increase the value of the parameter any further.

It is important to not take exceedingly high value for the parallelism in order not to saturate the dCache system, when running full scale jobs that could open connection in the order of the thousands.

For the globus-url-copy command, a similar study had to be carried out for the concurrency parameter, which represent the number of TCP connection to use simultaneously.

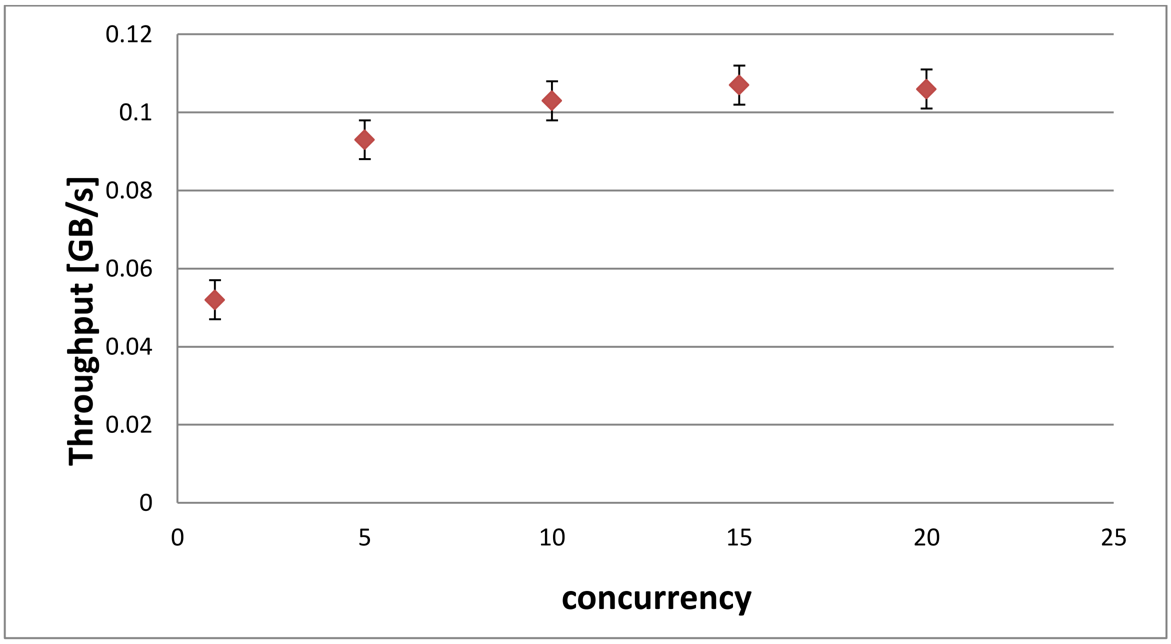

Figure 5: Study of the effect of the concurrency parameter over the total throughput

The results shown in Figure 5 are similar to those of Figure 4, and so is their analysis. It was determined that in order to maximize the throughput, while keeping the number of connections to a minimum, values of concurrency between 5 and 10 should be used.

During further tests with more VMs, it was shown that high values of concurrency and parallelism where causing the failure of 4 to 5% of the uploads, with no worthwhile increase in the total throughput, thus it was determined to use parallelism=4 and concurrency=5 for the study of the total throughput using more simultaneous VMs.

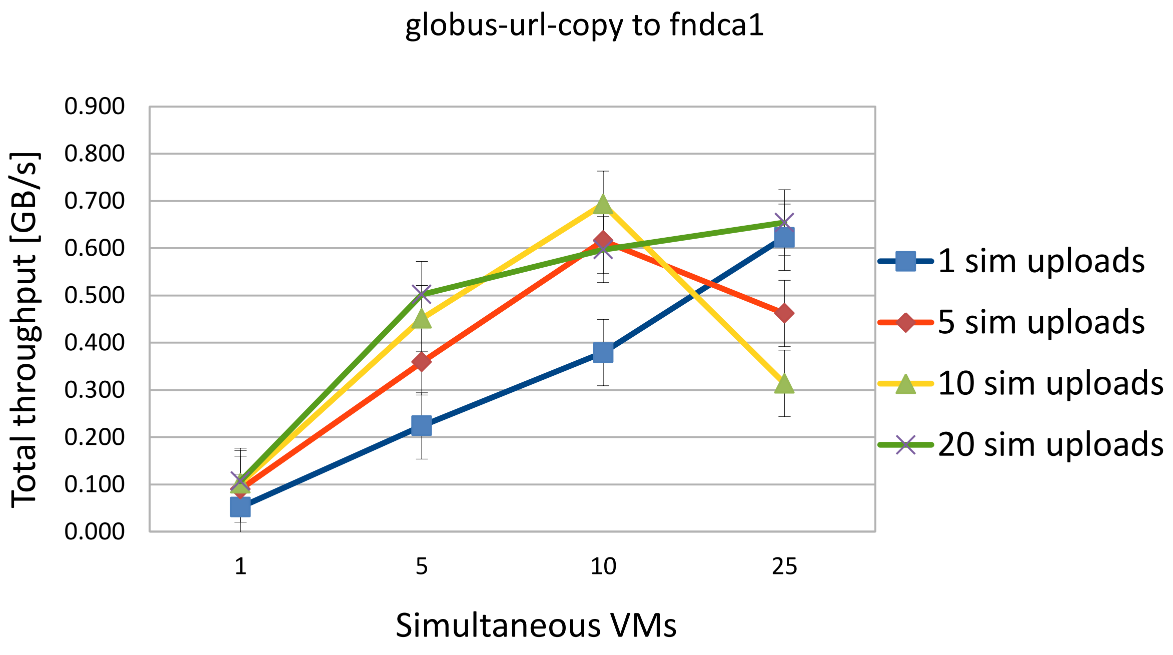

Figure 6: Total throughput analysis of the globus-url-copy command toward the fndca1 server

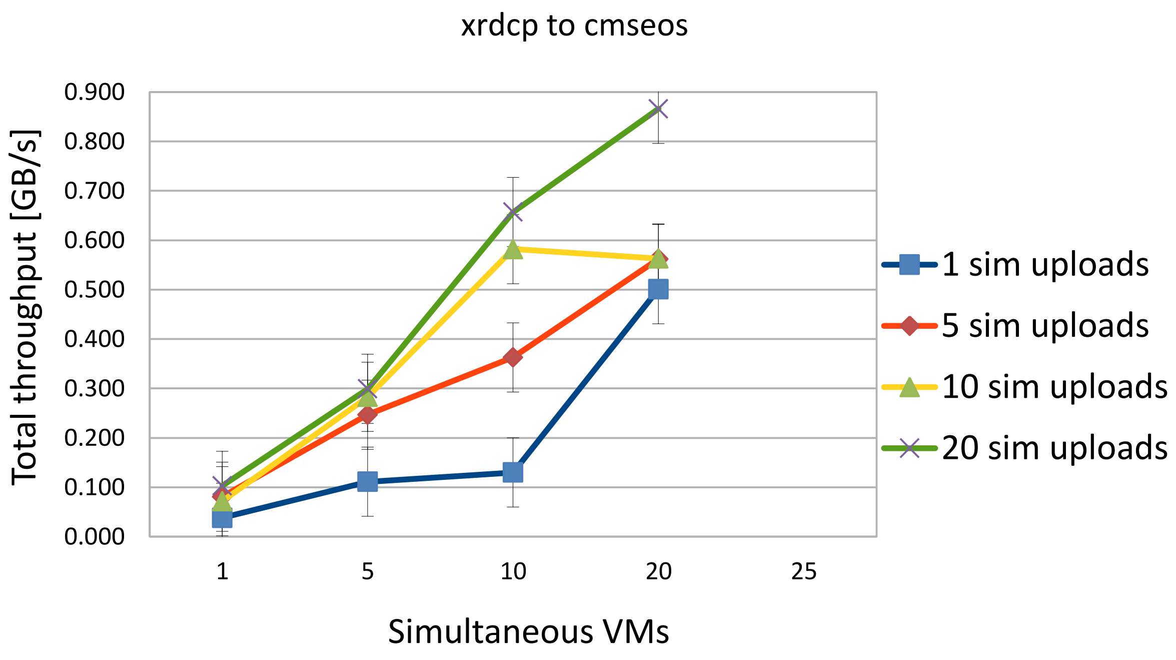

Figure 7: Total throughput analysis of the xrdcp command toward the cmseos server

The results shown in Figure 6 and Figure 7 demonstrate that a total bandwidth of 6 to 7 Gbit/s can be achieved, a value much higher than the 3Gbit/s require for a CMS job, that was calculated with the data available up to know.
This shows the capability of the system of handling the data transfer required, this satisfying one of the requests of the stakeholders.
Something that can be noted from the charts, is the unexpected decrease of the 10sim uploads throughput value when reaching 10 VMs. This is due to some of the uploads getting queued by the dCachce system and not executing in sync with the other, with an end result of a lower throughput for the 10 sim and an higher for the 20 sim. This effect was not eliminated, since the tasks of a CMS jobs will be launched sequentially, and it is expected that their data transfer won't be synchronized, but spread across the whole duration of the process.

6. Closing remarks

During the course of my work at Fermilab I acquired knowledge on cloud computing and its use in the scientific environment, and in particular I learned on how to work with the local FermiCloud and the public service of AWS. I also greatly improved my skill in writing bash scripts on a Linux environment and learned the usage of the AWS CLI and other homemade tools. I also gained knowledge on the usage of Kerberos certificates and identification, in order to set my credential or those of the VMs in the Fermilab environment.

Being my first working experience I also learnt how to be part of a group and carry out a work meant to be used in conjunction with the job of many other people.

During my stay, I was able to carry out all of the required jobs, by first running the gensim and hepspec06 benchmarks on public and local cloud resources, so to establish a metric to compare them in order to help decide the best solution for carrying out a CMS job. Afterwards I proceeded to write and execute bandwidth benchmarks, in order to assess if the capabilities of the system would fulfill the requirement of the stakeholder, and be able to run the jobs at maximum speed, with no delay due to the data transfer maximum throughput.

The data I acquired will have to be used in conjunction with the results from other studies of the spot market in order to decide the best solution for running the job with the best efficiency over cost ratio possible.

7. Acknowledgements

I want to thank my family that allowed me to be here at Fermilab by giving their support through all of the required processes.

I want to thank Simone Donati, Giorgio Bellettini and Emanuela Barzi, as the organizers of this Summer Internship and all of the Fermilab personnel that worked hard in order to allow me and the other summer students to have this wonderful experience.

I also want to thank the member of the computer division, and in particular my supervisor Gabriele Garzoglio and co-supervisor Steven Timm, for constantly guiding me through my work and the process of acquiring the knowledge to carry it out.