

Benchmarks of public and local cloud resources

Davide Grassano

Abstract – When acquiring computing power, either as bare metal machine or Virtual Machine, a metric has to be used in order to establish the most cost effective solution. To establish such metric, benchmarks have to be used in order to stress the same components that the final workload will be using. For a full scale CMS job the `ttbar_gensim` and `hepspec06` benchmarks were used to compare AWS instances with local machines in the FermiCloud. After establishing that the performance per core obtained were comparable and deciding the best instance to use, the study moved to the analysis of the bandwidth throughput of `c3.2xlarge` instances from and toward storage systems such as Amazon S3 or FermiGrid.

I. INTRODUCTION

Benchmarking is a commonly used technique to establish a metric, in order to compare the performance of machines with different architectures. It is common practice to use benchmarks when expanding the computing power of a facility, whether it be by buying bare metal machine, or by acquiring computing power from a third party on an on-demand basis.

The former generally requires the employment of generic and portable benchmarks, as to define the performance of the hardware at running a wide variety of tasks, while the latter calls for the use of specific benchmarks since the machine are bought only for the duration of a particular job, and should for this 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

II. EMPLOYED BENCHMARKS

Here is presented a brief description for the benchmarked used in this study.

A. `ttbar_gensim`

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%.

B. `hepspec06`

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.

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.

C. Bandwidth throughput tests

The bandwidth test 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.

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

[Type text]

the high latency from amazon to this storages, 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 IGB files were transfer simultaneously to the storage, from up to 25 VMs at the same time.

III. RESULTS

A. GENSIM AND HEPSPEC06

The results for the gensim and hepspec06 benchmarks can be found reported in Table 2.

| 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 2: Final results from the gensim and hepspec06 benchmarks on AWS instances

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, “based on the study [insert reference here]”.

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.

| 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 Opteron 2389 | 2.90 | 0.0217 | 0.174 | 12.0 | 96.1 |
| fnpc3000 | 16 | AMD Opteron 6134 | 2.30 | 0.0173 | 0.277 | 9.92 | 159 |
| fnpc4001 | 32 | AMD Opteron 6128 | 2.00 | 0.0149 | 0.477 | 8.64 | 277 |
| fnpc5009 | 32 | AMD Opteron 6134 | 2.30 | 0.0162 | 0.520 | 9.45 | 302 |
| fnpc6000 | 64 | AMD Opteron 6376 | 2.30 | 0.0136 | 0.873 | 10.0 | 640 |
| fnpc7024 | 64 | AMD Opteron 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 | | |
| cms wn2000 | 32 | AMD Opteron 6134 | 2.30 | 0.0167 | 0.5345 | 13.21 | 273.70 |

Table 1: Final results from the gensim and hepspec06 benchmarks on local Virtual and Bare Metal Machines

[Type text]

The results of the bandwidth analysis for reading from S3 are reported in Figure 1, 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.

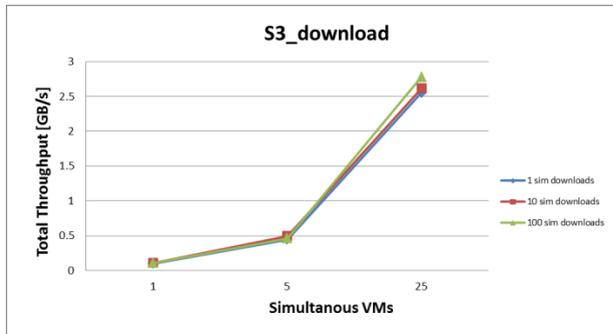


Figure 1: Download bandwidth throughput test from Amazon S3 to c3.2xlarge instances

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.

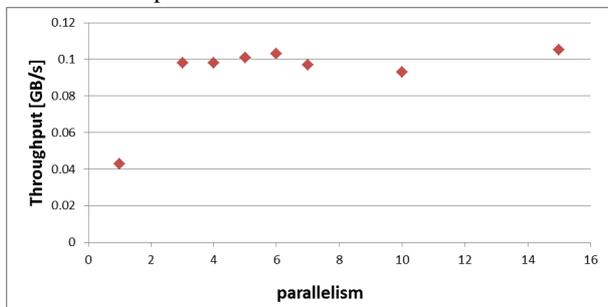


Figure 2: Study of the effect of the parallelism parameter over the total throughput

From the analysis of the data reported in Figure 2 and Figure 3 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.

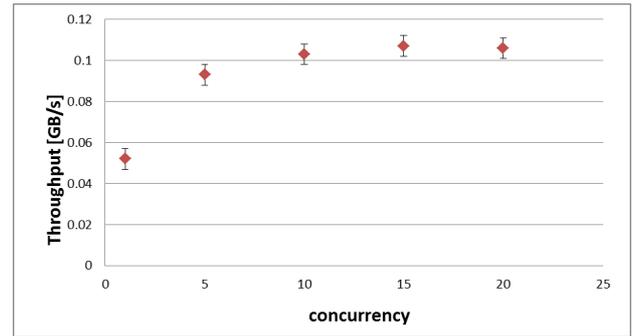


Figure 3: Study of the effect of the concurrency parameter over the total throughput

D. BANDWIDTH TEST TO FERMIGRID AND CMSEOS

Using the globus-url-copy command toward the fndca1 server, and the xrdcp command toward the cmseos server, the upload bandwidth throughput from c3.2xlarge instances toward FermiGrid was analyzed.

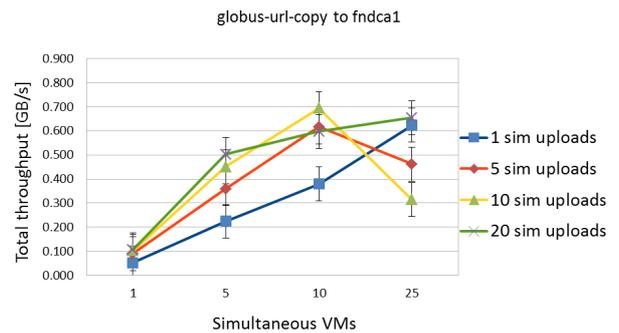


Figure 4: Total throughput analysis of the globus-url-copy command toward the fndca1 server

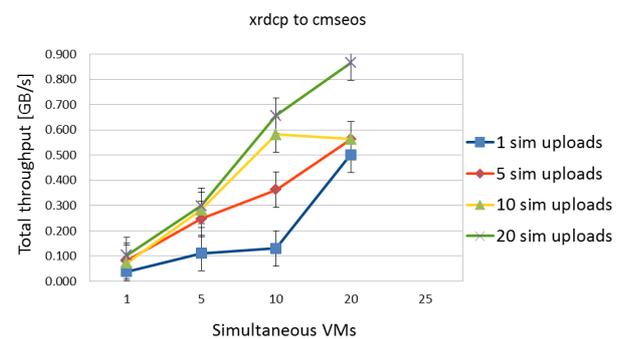


Figure 5: Total throughput analysis of the xrdcp command toward the cmseos server

The results reported in Figure 4 and Figure 5 shows 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. The second value being higher than

[Type text]

the first was an expected results, since the dCache set up on that server should be better than the other.

IV. CONCLUSIONS

Through these studies we were able to determine that the performances of public cloud resources are comparable to those of the local ones. It was also possible to obtain important data that will be used in order to determine the best solution for running a CMS job, after taking into account the analysis carried out over the spot market pricing and percentage of successful jobs.

If considering that a full CMS job will have 56000 core running, with each sending a 1GB file over the average of 8 hour, we have determined that the using c3.2xlarge instances toward FermiGrid, will give use almost 2.5x times the amount of required bandwidth, which demonstrates the fulfillment of one of the requests of the project stakeholders.