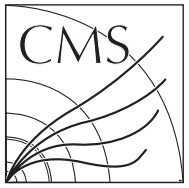
Available on CMS information server

CMS CR -2011/018



The Compact Muon Solenoid Experiment

Conference Report

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

11 January 2011 (v2, 03 February 2011)

Using Amazon's Elastic Compute Cloud to scale CMS' compute hardware dynamically.

Andrew Malone Melo for the CMS Collaboration

Abstract

Large international scientific collaborations such as the Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider have traditionally addressed their data reduction and analysis needs by building and maintaining dedicated computational infrastructure. Emerging cloud-computing services such as Amazon's Elastic Compute Cloud (EC2) offer short-term CPU and storage resources with costs based on usage. These services allow experiments to purchase computing resources as needed, without significant prior planning and without long term investments in facilities and their management. We have demonstrated that services such as EC2 can successfully be integrated into the production-computing model of CMS, and find that they work very well as worker nodes. The cost-structure and transient nature of EC2 services makes them inappropriate for some CMS production services and functions. We also found that the resources are not truely on-demand as limits and caps on usage are imposed. Our trial workflows allow us to make a cost comparison between EC2 resources and dedicated CMS resources at a University, and conclude that it is most cost effective to purchase dedicated resources for the base-line needs of experiments such as CMS. However, if the ability to use cloud-computing resources is built into an experiment's software framework before demand requires their use, cloud computing resources make sense for bursting during times when spikes in usage are required.

Presented at *CHEP2010: International Conference on Computing in High Energy and Nuclear Physics 2010*

Using Amazon's Elastic Compute Cloud to dynamically scale CMS computational resources

D Evans[1], I Fisk[1], B Holzman[1], A Melo[3,4], S Metson[2], R Pordes[1], P Sheldon[3], A Tiradani[1]

[1] Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, IL 60510, USA
[2] University of Bristol, Tyndall Avenue, Bristol BS8 ITL, UK
[3] Vanderbilt University, 6301 Stevenson Center, Nashville, TN 37235, USA

E-mail: andrew.m.melo@vanderbilt.edu

Abstract. Large international scientific collaborations such as the Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider have traditionally addressed their data reduction and analysis needs by building and maintaining dedicated computational infrastructure. Emerging cloud-computing services such as Amazon's Elastic Compute Cloud (EC2) offer short-term CPU and storage resources with costs based on usage. These services allow experiments to purchase computing resources as needed, without significant prior planning and without long term investments in facilities and their management. We have demonstrated that services such as EC2 can successfully be integrated into the production-computing model of CMS, and find that they work very well as worker nodes. The cost-structure and transient nature of EC2 services makes them inappropriate for some CMS production services and functions. We also found that the resources are not truely "on-demand" as limits and caps on usage are imposed. Our trial workflows allow us to make a cost comparison between EC2 resources and dedicated CMS resources at a University, and conclude that it is most cost effective to purchase dedicated resources for the "base-line" needs of experiments such as CMS. However, if the ability to use cloud-computing resources is built into an experiment's software framework before demand requires their use, cloud computing resources make sense for bursting during times when spikes in usage are required.

1. Introduction

Large international scientific collaborations face significant computing challenges: multi-Petabyte data sets that require petaflops of computing as well as globally distributed users and computing resources. To insure they meet their scientific goals, these collaborations have invested heavily in dedicated computing resources. These resources require expensive floor space, power, and cooling, as well as careful management, the use of grid technologies, highly developed strategies for data movement and placement, and the use of high performance networks.

The Compact Muon Solenoid[5] (CMS) experiment at the Large Hadron Collider[6] (LHC), for example, processes several petabytes of data yearly, using approximately 45,000 cores purchased and maintained by the experiment and spread globally across the grid. Most of the computing resources

[4] Corresponding author.
[5] http://cms.web.cern.ch/cms/
[6] http://public.web.cern.ch/public/en/LHC/LHC-en.html

owned by the experiment are needed on a continuous (24/7/365) basis and are used with relatively high efficiency. Their use is highly managed to insure that the highest priority experimental goals are met. There is some fluctuation in demand: the experiment doesn't log data year-round and user demand has some variation depending on the schedule of conferences. The experiment makes opportunistic use of shared resources such as those offered by the Open Science Grid[7]. Purchasing hardware to increase capacity is not cost effective if it will spend a significant fraction of its year unused: fixed costs and depreciation occur even if the machines are idle.

Recently, a number of "computing as a utility" or "cloud-computing" services have been started which offer users the ability to purchase CPU time at an hourly rate, with little to no commitment. Service providers are often internet businesses, such as Google and Amazon, that have had to build extremely large compute server facilities. Amazon's Elastic Compute Cloud[8] (EC2) is a commercial on-demand compute service where a user can rent virtual machines by the hour and merely pay the hourly rate plus bandwidth charges. The costs of housing, powering, cooling, maintaining, and managing this hardware is born by the provider. A number of users ranging from individuals and startup companies to large multinational corporations use EC2 and similar services to handle short-term computing needs, when it would be difficult, expensive and time-consuming to roll out more capacity for what may be a transient need[9].

2. Cloud Computing Services and CMS

Cloud computing resources free the user from the burden of housing, servicing, and managing expensive and fickle equipment. They also allow the user to purchase only those compute cycles and bits of storage needed to fulfill their specific goals or mission. In principle these can be purchased as and when needed. Economies of scale may lead to a lower support cost per unit of hardware, and these lower costs may be passed on to the user.

There are also potential downsides to cloud computing services. The provider must balance availability with inactivity – if it over buys hardware then it has increased its cost without recovery, and these costs must be passed on to the consumer. If it under buys hardware, then an important user advantage – being able to get access to resources when needed – is compromised. The provider must also make a profit, increasing cost. Economies of scale may not dramatically lower support costs, since large scale increases management and maintenance complexity, as does the need to support a user base with broad requirements and usage patterns. Supporting infrastructure that has only one "user" (like a dedicated CMS site) is likely to be significantly easier and less expensive.

We have investigated using EC2 to supplement or replace the compute capacity of CMS by renting out virtual machines and integrating them into our offline computing system. Our goal was to determine if this is both technically feasible and under what circumstances this makes financial sense.

3. Methodology

To explore the use of cloud computing services such as EC2 we first had to integrate EC2 resources into the CMS production framework. The framework requires the existence of permanent servers to manage the workflow, provide a condor master, provide database access, etc. It does not make sense to use cloud computing resources to provide these services – the pricing model is based on occasional use and cost recovery for these services is almost certainly based on a model that assumes nodes will not be used 100% of the time (24/7/365). We therefore decided to use EC2 nodes only as worker nodes in a larger production workflow. The architecture of this workflow is shown in figure 1. We had to figure out how to convince an EC2 node to behave just like a CMS worker node (see below), once that was done we were able to run sample production workflows and see how well the EC2 resources would

[7] http://www.opensciencegrid.org/
[8] http://aws.amazon.com/
[9] http://aws.amazon.com/solutions/case-studies/

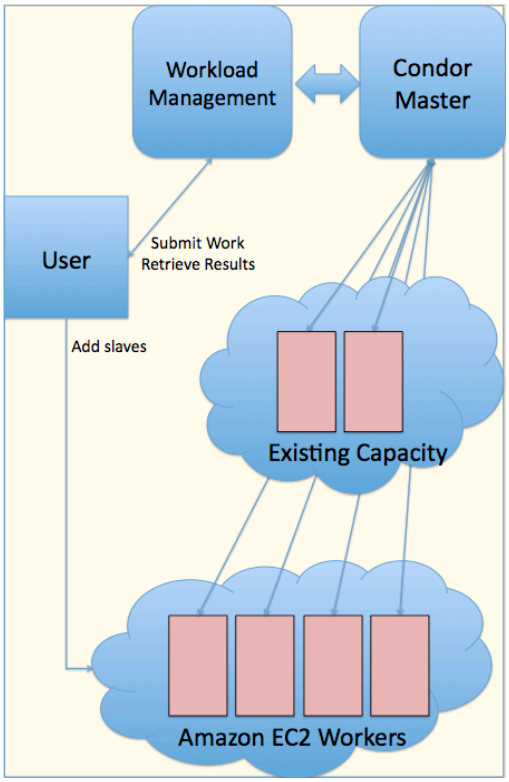


Figure 1. Architecture of the CMS production workflow. Once our VM is running on the EC2 instance, it is functionally equivalent to a regular CMS worker node.

work in the CMS production environment. We ran this workflow with (a) just EC2 worker nodes, (b) a mixture of EC2 nodes and regular CMS worker nodes, and (c) only regular CMS nodes at the beginning, to which we dynamically added EC2 nodes once the workflow had begun.

To convince an EC2 node to behave like a CMS worker node we put together a virtual machine (VM) slave that can receive work from an outside source, execute a CMSSW job to perform the work, and perform the necessary stage-out tasks on the output data.

Because of our requirement that the VM slave must have up-to-date versions of CMSSW installed, we decided to base it on the ready-built virtual machine provided by CERN (CernVM[10]). Maintained by CERN, it has a virtual, distributed filesystem that stores the CMSSW installs. This allows us to decouple the release schedule for our VM slave from the release schedule for CMSSW. An additional advantage of CernVM is that it comes with all the necessary CMS-specific prerequisites installed, some of which aren't provided in the SLC base install.

To CernVM we add a standalone condor slave. Our condor slave is a modified version of the condor glidein tool. Condor glidein allows you to insert "vanilla" cluster resources into a condor pool. For instance, if one had access to a non grid-aware cluster (running vanilla PBS lets say), submitting condor glideins to run on worker nodes of this vanilla cluster allows these nodes to function as members of a condor pool. The cluster will execute the glideins like a normal process; the glideins will boot and locate their masters, seek out work, and then execute jobs specified by the condor master (not the cluster master). One benefit of this infrastructure is that resources become homogenized, meaning the details of where and how jobs run are the same regardless of whether the hardware is local or running at EC2. This makes it possible to run workflows on a heterogeneous cluster, if needs demanded.

Architecturally, bootstrapping EC2 and connecting it to the CMS production framework (WMAgent) is simple. We wrote a wrapper around EC2's instance request tool to request an instance with CernVM's image and have it execute a small bootstrapping script on startup. This bootstrapping

[10] http://cernvm.cern.ch/

script downloaded a tarball with a condor startd daemon and some additional tools. Once the tarball was unpacked, a condor slave was executed which connected back to a central condor master, which makes the VM show up as a worker node to the production framework. We then had a WMAgent instance connected to the condor pool to create the jobs that would be run on EC2.

Our configuration also allows us to dynamically add new worker nodes to the pool, if needed. One could imagine a situation where, due to a deadline, some processing needed to be done soon, so the user could choose to add more nodes partway through their jobs. In our case, we were able to have a mixed cluster containing both Vanderbilt and Amazon resources by running condor slaves at each site. Once the slaves connected back to the master, they behaved similarly to each other, and our jobs could run without modification.

Early on, we made a choice to only test Monte Carlo event generation and not raw data reconstruction, which would have required bringing external event data into EC2. We had two reasons for this. First, EC2's bandwidth charges are prohibitively high and we predicted that these costs would've made an already cost-ineffective option worse. Additionally, we wanted to choose a CPU-bound test to make as equal of a comparison as possible since jobs running locally at Vanderbilt have the advantage of very fast, local worker node storage compared with EC2.

We generated a half million Monte Carlo events at both Amazon and Vanderbilt. At Amazon, the nodes were the "large" instance with 7.5GB of memory and 2 virtual cores. At Vanderbilt, we used 19 dual quad-core Opteron machines with 32GB of ram each. We also tested a configuration where we ran with a fixed number of cores at Vanderbilt and later added additional cores to the pool from EC2. Finally, we ran the same Monte Carlo generation procedures on the differently sized EC2 instances to see if the performance scaled with size.

Once we connected all the components together, we generated 500,000 events on both EC2 and at Vanderbilt using the same configuration to make a comparison between EC2 and a dedicated cluster. We then benchmarked how long it took for each cluster to produce the events.

4. Results

One unanticipated lesson we learned from this activity was that EC2 was not fully an "on-demand" service. Amazon initially placed a very low cap on the number of processors we could use at any one time. It took a day or so to get this cap increased, and it took a few iterations of this procedure to get up to the "large" number of processors (a few hundred) we required for our sample production workflows. We do not know how difficult it would be to get a few thousand cores, but based on our interactions our guess is this would take some negotiation and planning with Amazon. This is almost certainly going to be true for the cloud computing services offered by other vendors. It is probably true that as one established a relationship with vendors, this would become a simpler and more rapid process. However, one could also imagine a scenario where a large number of additional cores were needed quickly, and a vendor would not have those resources available on a sufficient timescale.

The cloud computing model assumes that the services provided will be put up and torn down as they are needed, and cost recovery for these services is almost certainly based on a model that assumes nodes will not be used 100% of the time (24/7/365). In addition, local control and administration of some systems will always be necessary. Therefore, investment by experiments in dedicated hardware will always be necessary. CMS will always need to invest in some dedicated computing hardware, since some servers and services need to be owned, managed, and administered by the experiment and some services need to be provided 24/7/365.

Using EC2 cores to provide additional or even all worker nodes for a production workflow worked extremely well. We did not test using more than two hundred cores at a time, but we saw no technical reason why usage couldn't successfully scale to even larger numbers of cores. We purposely chose a CPU-bound test and, including the overhead from staging the output file from EC2 to Vanderbilt's Storage Element (SE), achieved approximately 85% the performance our local cluster achieved. Though we had to transfer the output files over the internet to Vanderbilt's storage element, this overhead only contributed a few minutes to the 6-8 hours that each job ran on an EC2 instance. We

attribute most of this difference to differing speeds of the CPU cores at the two sites; we saw no evidence that there was any difference in "efficiency" (CPU time divided by clock time) for our jobs at either site and the overhead was a relatively small contribution. Once we assembled the software necessary to successfully start a worker process on an EC2 core, these "worker nodes" proved as easy to use as those at dedicated CMS sites. We were even able to dynamically add EC2 cores to the pool of worker nodes that were working on a production workflow, allowing us to scale up the processing power for a workflow on the fly.

One of our goals was to compare the cost of on-demand cloud-computing services to dedicated hardware housed and managed by the experiment. From our tests, we are able to make a direct comparison of the costs of time on EC2 to the costs of a dedicated facility[11] housed and operated at Vanderbilt University. In our CPU bound tests, generating 1000 Monte Carlo events at EC2 cost us $0.817 (US $) for CPU time (computed from the per-hour rate of $0.18/core) and $0.055 for bandwidth (moving the generated data out of EC2 and into CMS owned storage), for a total cost of $0.872 per 1000 events. The Vanderbilt site provides dedicated resources at a cost of $213.60 per core per year. This amortizes the purchase price of the hardware over three years, and includes all power, cooling, facilities, network bandwidth, and the costs of support, maintenance, and administration. The hardware includes the compute node and an appropriate share of the costs for internal cluster networking, racks, cluster system services, and shared disk space. One core on the Vanderbilt cluster could generate 244 Monte Carlo events an hour in our sample workflow, including all overheads. Assuming that a core is generating events full time, this means that the cost of generating 1000 events on the dedicated resources at Vanderbilt is $0.098, a factor of almost 9 less. Of course the cost of these dedicated facilities increases if they are not used 100% of the time.

5. Conclusions

We have demonstrated that services such as EC2 can successfully be integrated into the production-computing model of CMS. We have carried out example production runs that incorporate cloud-computing resources as worker nodes in a production workflow, under this usage scenario they work very well.

We used both EC2 and Vanderbilt to generate a half million Monte Carlo events. We purposely chose a CPU-bound test and found that the EC2 cores performed well as worker nodes. In addition, we demonstrated that we could dynamically add EC2 nodes to an already instantiated production run, which would allow a high priority production workflow to complete earlier if required.

We found that the cost-structure and transient nature of EC2 services makes them inappropriate for some CMS production services and functions. We also found that the resources are not entirely "on-demand." Limits and caps on usage occur. Based on our cost comparison between EC2 resources and dedicated CMS resources at Vanderbilt University, we conclude that it is most cost effective to purchase dedicated resources for the "base-line" needs of experiments such as CMS. However, if the ability to use cloud-computing resources is built into an experiment's software framework before demand requires their use, we conclude that cloud computing can be a useful tool for bursting during times when spikes in usage are required.

The source code for this project is available at http://bit.ly/cRbwHv.

Acknowledgements

The authors would like to thank Lothar Bauerdick for helpful discussions, support, and guidance. We would also like to thank Predrag Buncic and the CernVM team for providing the necessary hooks into CernVM for on-boot contextualization.

[11] ACCRE, the Advanced Computing Center for Research and Education, http://www.accre.vanderbilt.edu/.