Managing competing elastic Grid and Cloud scientific computing applications using OpenNebula

Journal of Physics: Conference Series **664** (2015) 022004 doi:10.1088/1742-6596/664/2/022004

Managing competing elastic Grid and Cloud scientific computing applications using OpenNebula

S Bagnasco[1], D Berzano[2], S Lusso[1], M Masera[1,3], S Vallero[1,3]

[1] Istituto Nazionale di Fisica Nucleare, Via Pietro Giuria 1, 10125 Torino, IT
[2] CERN - European Organization for Nuclear Research, CH-1211 Geneva 23, CH
[3] Dipartimento di Fisica, Università degli Studi di Torino, Via Pietro Giuria 1, 10125 Torino, IT

E-mail: stefano.bagnasco@to.infn.it

Abstract. Elastic cloud computing applications, i.e. applications that automatically scale according to computing needs, work on the ideal assumption of infinite resources. While large public cloud infrastructures may be a reasonable approximation of this condition, scientific computing centres like WLCG Grid sites usually work in a saturated regime, in which applications compete for scarce resources through queues, priorities and scheduling policies, and keeping a fraction of the computing cores idle to allow for headroom is usually not an option. In our particular environment one of the applications (a WLCG Tier-2 Grid site) is much larger than all the others and cannot autoscale easily. Nevertheless, other smaller applications can benefit of automatic elasticity; the implementation of this property in our infrastructure, based on the OpenNebula cloud stack, will be described and the very first operational experiences with a small number of strategies for timely allocation and release of resources will be discussed.

1. Introduction

Applying Cloud Computing technologies to scientific computing has been investigated ever since the public availability of Amazon's EC2 service in 2006, and even earlier following the availability of virtualization technologies for resource optimization. Research on the topic still follows two paths:

- porting scientific High Throughput or even High Performance computing workloads to commercial public clouds;
- building private clouds specifically tailored for scientific computing applications.

In this work we will focus on the latter.

Both activities, in the High Energy Physics community at least, can be seen as stemming from the Grid Computing era, in which the concepts of pooling and sharing resources did not arrive as far as actually providing on-demand computing power in such a simple and transparent way as the "power grid" metaphor [1] promised. The WLCG [2], for example, has been highly successful in providing efficient computational and storage resources to the LHC experiments and the HEP community at large, but is not much more than a very large, sophisticated, highly distributed batch system that spans several administrative domains and serves multiple Virtual Organizations. For a number of reasons [3], the Grid Computing paradigm was never widely adopted outside an enclave in the scientific computing community; the Cloud Computing concept, which (simplifying somehow) adds

virtualization technologies to the Grid concept, arguably helped develop a computing-as-a-service model that proved to be attractive also for commercial providers.

1.1. Cloud computing for science

As far as the user is concerned, resources in commercial public cloud computing infrastructures are virtually infinite: provided she's prepared to pay for them, according to some pricing policy, any number of machines she instantiates run almost immediately. This is made possible by a number of factors. First, many applications do not require high throughput but scalability and resiliency of a number of interacting services that can scale out (think for example of an e-commerce application, with web servers, DB instances, backoffice applications,…) so resources can often be effectively overcommitted. Second, most public cloud infrastructures are very large compared to any single application running on them; this, coupled to the fact that most commercial applications use relatively small virtual machines, makes optimisations easier. Last, the business model itself is based on providing large infrastructures that have to be used efficiently but can grow physically with demand (see, for example, the discussion in [4]).

At the opposite end, most scientific computing infrastructures such as Grid sites almost always run in a saturated regime: resources are limited, or in any case finite, while computational needs are potentially infinite. Given sufficient time to develop their applications, researchers can find unsolved problems to saturate any amount of resources. Furthermore, many scientific applications scale more optimally up than out, so virtual machines used for such use cases tend to be rather large: a smaller number of large machines may be more efficient, from the purely computational point of view, of several small ones.

On top of this, the model used by many funding agencies is still to have the experiment collaboration evaluate their needs for the next period, expressing them in some units like the widely used HEPSpec06 benchmark results [6], give them to a college of referees to evaluate and then pay for a given amount of, for example, CPUs that will be bought and used. Even though in the Grid model the resources are in principle first pooled then shared, they are still seen as proprietary. As a consequence, economical scheduling implying (virtual) billing of consumed resources has never been widely adopted (for an introductory discussion of such principles, see again, for example, [3]); in other words, resource allocation is made in HEPSpec06 units instead of HepSpec06×hour. This translates into an issue in deploying real elastic applications where the amount of used resources is calculated (and billed) *a posteriori* and not just audited.

2. Cloud computing at INFN-Torino

The Computer Centre at INFN-Torino is a medium-size infrastructure providing computing and storage resources to a number of scientific computing applications, the largest ones being a WLCG Tier-2 site for the ALICE experiment at the CERN LHC, a PROOF-based Analysis Facility for the same experiment [8] and a separate Grid Tier-2 centre for the BES-III Experiment at IHEP, Beijing [9]. Moreover, it provides computing resources to some smaller communities such as, for example, a local theoretical physics group or a collaboration developing on-demand medical imaging tools.

The site hosts about 1.6 PB of disk storage and approximately 1300 computing cores; the Cloud infrastructure, managed with version 4.8 of the OpenNebula stack [10], is constantly growing and currently comprises more than 70 hypervisors and hosts about 250-300 virtual machines.

Historically, the decision to adopt a Cloud infrastructure to provision resources was taken to allow the centre to dynamically reallocate resources between the two ALICE applications, one a traditional Grid batch farm, the other a Parallel Interactive facility that provided rapid turn-around computing power to small use cases for which the Grid presented too large an overhead to be practical. This was furthered by gradually moving all services needed to run the WLCG Grid site onto the IaaS platform, and subsequently by adding more tenants to our infrastructure (for a complete description, see [11]).

The obvious next step was to provide some of the applications with automatic elasticity, i.e. the capability to automatically (autonomously or by means of a dedicated service) start or stop their Virtual Machines according to their need.

This needs two aspects to be taken care of: the tools to monitor the application's needs and start or stop VMs accordingly, and a way to enforce policies and priorities.

3. Two paths to elasticity

Many real-world applications in our environment are not intrinsically elastic. Missing a billing system, users have no incentive to release their resources even if they are unused, and once a VM is instantiated by a given tenant, it needs a fair amount of persuasion to have it undeployed. This is an instance of sorts of the "tragedy of the Commons" famously described by G. Hardin in 1968 [12]: a free common resource tends to be depleted to waste by independent users that act rationally and autonomously, like sheep belonging to different herds grazing on the same public patch of grassland. Since the grass is free, if the shepherds act independently of each other the most rational strategy *for each of them* appears to be to increment the number of sheep, until the resource is thrashed. So, quite obviously, some incentive mechanism (e.g. billing, a fairshare mechanism, dynamic quotas…) needs to be in place for the system to work at all. For a description of ongoing work on the topic in our group, see [13].

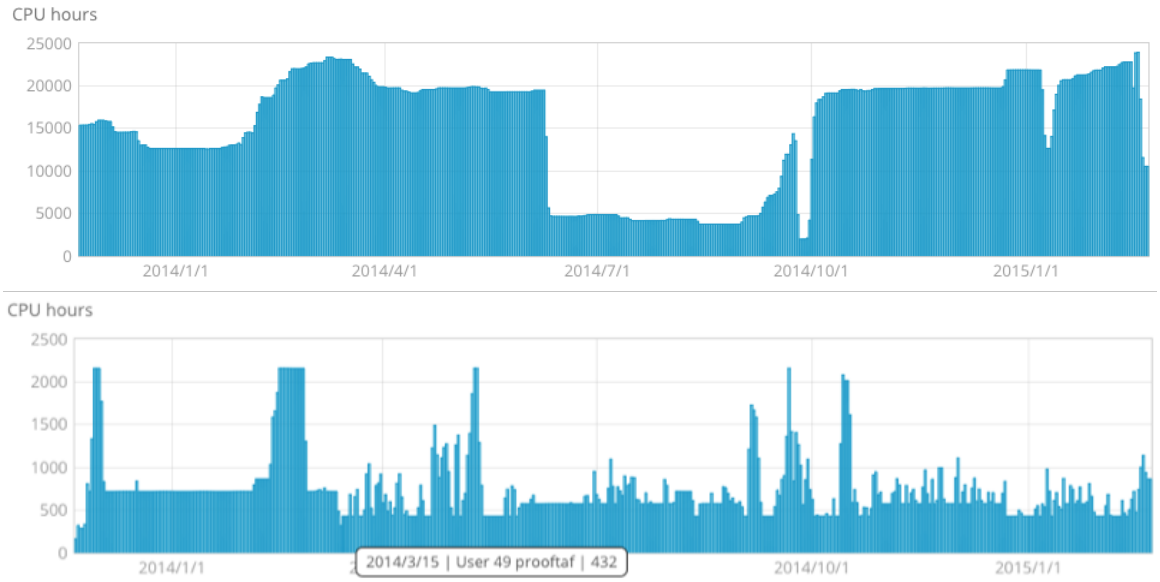

Figure 1. Usage patterns for Grid nodes (top) and VAF nodes (bottom).

We explored two different ways of providing automatic elasticity to our application. The first is completely generic, relying on an external tool; the second, that seemed promising but proved itself only partially effective without some further work, used the orchestration component of OpenNebula. The applications on which we experimented such ideas are the ones that had non-continuous usage patterns, i.e. the Virtual Analysis Facility (that being interactive exhibits clear daytime and weekday usage patterns), the BES-III Grid Tier-2 site (which is currently not always in full use) and our standard On-demand Virtual Farms, simple batch farms that are used by some tenants with simpler use cases.

3.1. elastiq

elastiq [14] is a custom Python daemon originally developed for the Virtual Analysis Facility, but being independent of PROOF can be used by many other applications. It uses the EC2 interface of the

underlying Cloud controller (in our case OpenNebula) to monitor, deploy and stop virtual machines, monitoring a simple queue (the implemented plugin uses HTCondor [15]) to assess the application's need. The user deploys by herself the single master Virtual Machine, specifying in the context the worker configurations. The application is then automatically scaled up and down by elastiq by starting more VMs when there are "jobs" in the queue, or when nodes are idle.

The usage patterns for a static application like the Grid site and the elastic Virtual Analysis Facility are shown in figure 1: the VAF shows spikes of activity only when it is used, while the peaks and troughs of the Grid usage is dominated by resource availability. Besides the Virtual Analysis Facility, we use elastiq to provide automatic scalability to some other on-demand elastic farms.

3.2. OneFlow

We also tested the opportunity of using the OpenNebula orchestration component, OneFlow [16], which can be used to deploy clusters of Virtual Machines, expressing dependencies between them. This offers several advantages: there are no external tools to integrate and maintain, it is very simple to configure a full cluster as a single service, and it can autonomously scale up and down the applications.

One possible use case was the BES-III Tier-2 grid site, given that at the moment the flow of jobs to be executed there is intermittent (as opposed, as we will see in the next section, to the WLCG Tier-2 site). We tried deploying an instance through OneFlow, by describing a CREAM CE [17] as master node and a variable number of DIRAC [18] Grid WNs as slaves, with the usual problem that the LRMS is PBS, which is not cloud-aware. The Worker Nodes publish the number of queued and running jobs to OneGate, the OpenNebula component designed to gather custom metrics from VMs [19]. The system is thus able to scale up one VM at a time when there are queued jobs.

Figure 2 shows the CPU load of three BES-III Worker Nodes under the control of OneFlow. Job submissions trigger the deployment of new machines, while only at the end of all jobs the machines can be turned off. This is the drawback of such approach: OneFlow is primarily designed for load-balancing applications. In load-balancing applications, the nodes (web servers, for example) are stateless and more can be deployed to handle some of the queries should they become overloaded. Conversely, if the overall load allows it, it is possible to turn down any of the servers and release the resources, since the remaining nodes will go on processing queries. In our case, however, it is not the overall load that is important but the load on specific machines, and since with OneFlow it is not currently possible to choose which nodes to undeploy, the only available strategy (without further implementation work) is to scale down the full infrastructure when all the jobs are finished, which is clearly sub-optimal.

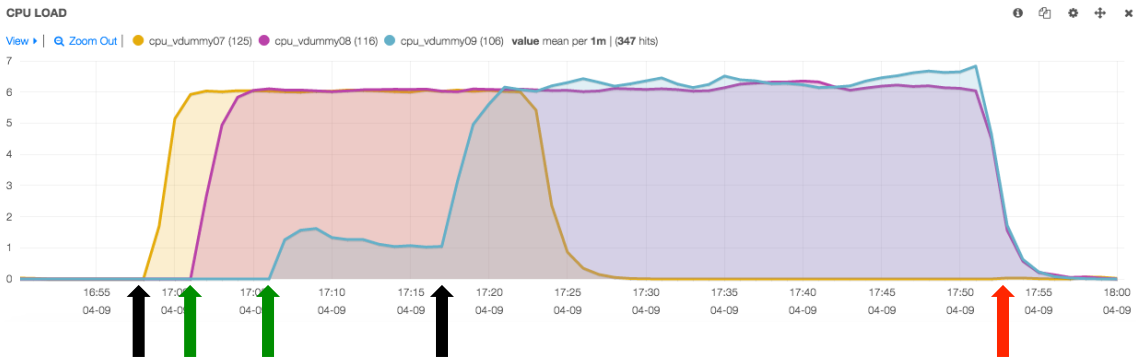

Figure 2. CPU load for three Grid WNs controlled by OneFlow.
Black arrows: job submission; green arrows: scale up; red arrows: scale down.

4. The inelastic wall challenge

It should now be noted that the largest application in our site, the Grid Tier-2, is in practice completely inelastic. The resources are almost always saturated (see figure 3), so in the assumption that other workloads may have a higher priority, a policy needs to be devised to stop the WNs to make room for other applications. In the following, we describe some problems that need to be addressed and a possible solution.

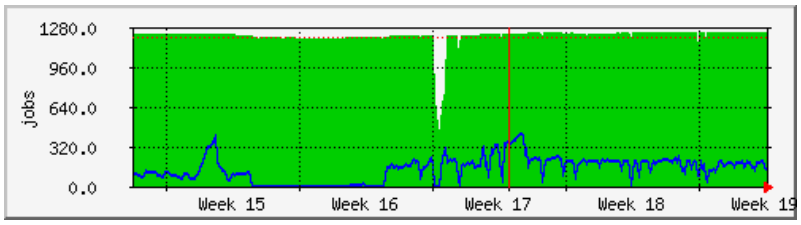

Figure 3. One month of typical occupancy of the Grid Tier-2 queues, all VOs. Green: running jobs, blue: queued jobs.

Usually, the way of representing competing applications in a scientific computing system is picturing it as a batch system of sorts, with different queues given appropriate priorities and a scheduler doing its job to enforce policies, for example fair share. However, in this kind of situation it may be better to picture the competing use cases as long-running applications that can elastically grow and shrink according to needs and the amount of available resources, which are always fully busy.

In our particular environment, however, the inelastic Grid Tier-2 site is much larger an application than all the others. A possible strategy is thus to artificially lower its Young modulus, making it shrink and release resources when smaller, higher priority ones "push" and require them. Then the other elastic applications will compete for the available resources: each application will have a Young modulus assigned (the higher the priority the stiffer the application) and apply a pressure proportional to its needs. It should here be noted that the assumption that Grid jobs have a lower priorities than others is somewhat justified by the fact that there's a virtually infinite supply of them (so in principle it can apply infinite pressure).

So a possible strategy would be to periodically drain a number of Worker Nodes (Grid jobs are seldom, if ever, checkpointable) and check whether there are other applications needing resources. Should this be the case, the system would kill the WN and start the application's VMs, otherwise the WN can be restarted. In this schema, management of the WN can be done by several means, from simple schedulers like the ones included in Cloud controller stacks to pilot factories like vCycle [20].

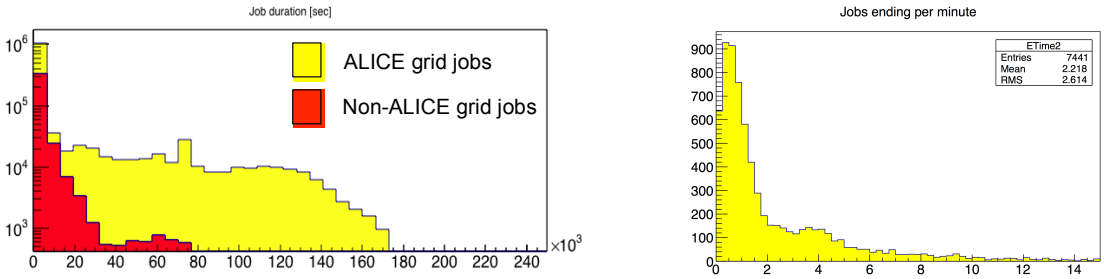

Figure 4. Job duration distribution (in seconds) of Grid jobs (left), number of jobs terminating per minute (right).

However, a number of problems arise.

- The IaaS infrastructure has, by design, no knowledge of what's happening inside the Grid Worker Nodes running on its hosts, and it would be complex for an external daemon to gather information about Grid jobs. For the same reason, it is very difficult to optimize job distribution across worker nodes (e.g. packing same-length jobs on the same WN to have them end at the same time). Furthermore, the job duration distribution shows no obvious pattern (see figure 4, left), so it provides no easy criteria for choosing which multicore WN to undeploy.
- We use rather basic OS images that are contextualized at deploy time. The complex operations needed to fully configure a WLCG WN mean it can take up to 20 minutes from the deployment of the image on the hypervisor to the moment the node starts accepting jobs. Furthermore, the complex dependencies of Grid middleware mean that without special measures, WNs configured at different times may come out with different sets of packages, or not work at all. This can be mitigated by using special pre-configured images for WNs.
- To reduce the overall number of virtual machines and the associated overhead, we run rather large (6 or 8 cores) virtual Worker Nodes. This means that to undeploy a WN, one needs to wait for 6 to 8 jobs to finish, which potentially means up to a few days in which the already idle CPUs are wasted. Single or dual-slot WNs are more effective; to get a qualitative idea of the latencies, one can look at the distribution of the number of jobs finishing in any given minute (see figure 4, right); the average value over one year of operation is 2.2, with a mode of about 1. However, there are drawbacks: more disk space on hypervisors, more CPU and memory needed for virtualization, more administrative overhead.
- Many LRMSs used in Grid sites (e.g. the ones from the OpenPBS family, like Torque [21]) are not cloud-aware and cannot easily cope with worker nodes appearing and disappearing from their clusters. This was mitigated by means of custom scripts that take care of updating the nodes list and restarting the relevant daemons on the server when needed, but other systems (like the widely used HTCondor) are better suited to this condition.

We will implement such measures and test the ideas in the near future. In order to balance between having a reasonably small number of virtual machines and making small worker nodes that can be drained quickly, we plan to use an admixture of large (6-8 cores) and very small (1-2 cores) WN, the latter managed by a separate, dedicated CE running HTCondor that will be dynamically scaled either by elastiq or by an adapted version of OneFlow. It will then be possible to test more ideas: the tuning of virtual Worker Node parameters (number of cores and lifetime) to match the statistical distribution of job durations, or the balance between the amount of resources statically pinned to an application and the amount left available for competitive seizing by elastic applications.

5. Conclusions and outlook

Bringing automatic elasticity to scientific computing Cloud infrastructures, always working with saturated resources, poses several problems. The opportunity of having a much larger applications allows us to experiment with the strategy of having it passively shrink to make room for others, even though having this tuned will need more work.

One of the tasks of the recently started EU project INDIGO-Datacloud [22] is indeed the development of an advanced VM scheduling service to allow using of Cloud Computing infrastructures to provide computational power to several application competing for them in a saturated regime, as such is the case of most scientific computing use cases. Our group is involved in that activity, that we see as a natural application of some of the ideas presented in this paper.

Acknowledgements
The present work is supported by the Istituto Nazionale di Fisica Nucleare (INFN) of Italy and is partially funded under contract 20108T4XTM of Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale (PRIN, Italy).

References
[1] Foster I and Kesselman C 1998 *The grid: blueprint for a new computing infrastructure* (San Francisco, CA: Morgan Kaufmann)
[2] http://wlcg.web.cern.ch/
[3] Sandholm T and Lee D 2014 arXiv:1504.07325v1
[4] Erl T, Zaigham M and Puttini R 2013 *Cloud Computing – Concepts, Technology & Architecture* (Upper Saddle River, NJ: Prentice Hall)
[5] http://www.gridpp.ac.uk/vcycle/
[6] http://w3.hepix.org/benchmarks
[7] Aamodt K *et al*. (The ALICE Collaboration) 2008 *JINST* **3** S08002
[8] Berzano D, Bagnasco S, Brunetti R and Lusso S 2012 *J. Phys.: Conf. Ser.* **368** 012019
Berzano D, Blomer J, Buncic P, Charalampidis I, Ganis G, Lestaris G and Meusel R 2014 *J. Phys.: Conf. Ser.* **513** 032007
[9] Ablikim M *et al*. 2010 *Nucl. Instr. Meth. A* **614** 345–399
[10] Moreno-Vozmediano R, Montero R S and Llorente I M 2012 *IEEE Computer* **45** 65-72
[11] Bagnasco S, Berzano D, Brunetti R, Lusso S and Vallero S 2014 *J. Phys.: Conf. Ser.* **513** 032100
[12] Hardin G 1968 *Science* **162**(3859) 1243–1248
[13] Bagnasco S, Berzano D, Guarise A, Lusso S, Masera M and Vallero S 2015 *Integrated Monitoring-as-a-service for Scientific Computing Cloud applications using the ElasticSearch ecosystem*, in these proceedings
[14] https://github.com/dberzano/elastiq
[15] Thain D, Tannenbaum T, and Livny M 2005 *Concurrency and Computation: Practice and Experience*, Vol. **17**(2-4) 323–356
[16] http://docs.opennebula.org/4.8/advanced_administration/application_flow_and_auto-scaling
[17] Andreetto P *et al*. 2011 *J. Phys.: Conf. Ser.* **331** 062024
[18] Fifield T, Carmona A, Casajùs A, Graciani R and Sevior M 2011 *J. Phys.: Conf. Ser.* **331** 062009
[19] http://archives.opennebula.org/documentation:rel4.4:onegate_usage
[20] http://www.gridpp.ac.uk/vcycle/
[21] http://www.adaptivecomputing.com/products/open-source/torque/
[22] http://www.indigo-datacloud.eu