Dynamic VM Provisioning for TORQUE in a Cloud Environment

S. Zhang¹, L. Boland², P. Coddington¹ and M. Sevior²

¹eResearch SA, Adelaide, South Australia, Australia  
²The University of Melbourne, Melbourne, Victoria, Australia

E-mail: shunde.zhang@ersa.edu.au

Abstract. Cloud computing, also known as an Infrastructure-as-a-Service (IaaS), is attracting more interest from the commercial and educational sectors as a way to provide cost-effective computational infrastructure. It is an ideal platform for researchers who must share common resources but need to be able to scale up to massive computational requirements for specific periods of time. This paper presents the tools and techniques developed to allow the open source TORQUE distributed resource manager and Maui cluster scheduler to dynamically integrate OpenStack cloud resources into existing high throughput computing clusters.

1. Background

Cloud computing promises governments and enterprises many advantages over more traditional computing models. More efficient resource sharing, improved manageability, less maintenance and dynamic resource reallocation to name but a few. These attractive properties have led to the proliferation of cloud tools and services from both commercial and government agencies alike and has changed the way in which computing resource management is being viewed.

The more traditional users of large scale computing resources, for example the high throughput computing (HTC) facilities used by the high energy physics (HEP) community, have previously developed and used technology on dedicated computing clusters that, although very efficient in resource utilization and job scheduling, have not had the same flexibility that cloud computing offers. As managers and funding agencies strive to get the best computing performance for each dollar spent it is worth investigating ways to augment the dedicated HTC clusters with the available cloud computing resources on both a needs basis or as an opportunistic deployment of under utilized shared cloud resources. This is particularly attractive to HTC workloads that are bursty in nature since it would dynamically grow the cluster into the cloud during busy times and retreat when not needed.

An example of the interest and investment in cloud computing is the Australian federal government’s National eResearch Collaboration Tools and Resources (NeCTAR) project [1] which has built a national Research Cloud for Australian researchers based on the OpenStack cloud computing software. The Research Cloud consists of 5,000 computing cores geographically spread across the whole continent at 3 hosting nodes and presented as a single federated cloud system. It will be expanded to over 30,000 cores at 8 hosting nodes next year. Individual researchers and research centres can apply for computing resources to meet their computational needs and computing core allocations are assigned based on national research priorities.

The Terascale Open-Source Resource QUEue Manager (TORQUE) [2] and the Maui [3] job scheduler are a popular open-source solution for controlling HTC clusters and are widely used in the
Australian HEP environment. Australian researchers are familiar with its job submission language and the cluster administrators are skilled in its configuration and tuning. Although many other batch systems exist, some already with more advanced cloud utilization features, the familiarity and experience with TORQUE and its compatibility with grid infrastructure meant work on integrating it with OpenStack would be beneficial for the Australian HEP community.

2. Related work
As interest in cloud computing has grown so has the effort of integrating HTC environments to make use of cloud services. With numerous competing cloud technologies emerging and the existence of many batch queuing systems there are also many unique and specialized solutions being actively developed. The following section presents a selection of the work that most influenced the tools developed to integrate TORQUE with OpenStack.

The grid-middleware project: “Distributed Infrastructure with Remote Agent Control” (DIRAC [4]) developed the capability to submit jobs to EC2 compatible IaaS clouds via an extension to its pilot software agents. Virtual machines (VMs) were deployed on the IaaS clouds with the pilot agent pre-installed in the OS. Upon instantiation, agents contact the central DIRAC server and request payload jobs which were delivered to the VM. DIRAC developed 3 additional services to enable this feature: A Virtual Machine Scheduler which monitors DIRAC TaskQueues and requests new VM’s from resource provider as appropriate. A Virtual Machine monitor which is an on-VM module that reports activity and halts the VM if no longer needed and a Virtual Machine Manager which collects information about requested, running and halted VMs. It also provides usage monitoring with functionality accessible as an extension to the DIRAC web interface. The DIRAC system was used to control a 800-core collection of VM’s in Amazon EC2 IaaS Cloud [5]. However the ATLAS experiment does not use the DIRAC pilot factory and so it could not be employed for all our use cases. Moreover Torque is a well supported back end for the CREAM CE which is widely used and well maintained by EGI.

Cloud Scheduler [6] is a suite of python scripts that run alongside Condor [7]. Condor is a resource manager that specializes in HTC jobs. Cloud Scheduler monitors Condor’s job queue, if new jobs are submitted to Condor and there are not enough worker nodes in Condor, Cloud Scheduler launches new VMs in the cloud and adds them to the Condor pool, up to a specified limit. These jobs will then be dispatched to the newly-added worker nodes via Condor’s built-in mechanism. Once jobs are finished, if there are no more jobs in the Condor queue that can use them, Cloud Scheduler shuts down idle worker nodes, and returns the VMs to the cloud. Currently Cloud Scheduler is being used by some members of the ATLAS experiment to analyze data collected by the Large Hadron Collider at CERN, and we have successfully used it to run ATLAS test jobs on the NeCTAR cloud.

StarCluster [8] is an open source cluster toolkit developed by MIT for Amazon’s EC2. It has been designed to automate and simplify the process of building, configuring and managing clusters of VMs for various purposes by using different plugins. Currently, there are not only plugins for resource management systems for computing clusters, such as SGE and Condor, but also plugins to build MySQL clusters and Hadoop. StarCluster builds a series of VMs and configure them as a cluster, including one head node and a number of worker nodes. The head node runs a shared file system for all other worker nodes to access. The chosen plugin installs the corresponding resource management system on all these VMs and make it ready for the user to use. The user only needs to log in to the head node to submit batch jobs, which will be distributed to all other worker nodes for execution. It enables users to create private clusters in the cloud. However, because StarCluster is designed only for EC2, it uses a few EC2-specific features, which makes it unable to be used on OpenStack. StarCluster did not have a plugin for TORQUE either.

ViBatch [9] makes use of a local virtualization hypervisor, such as KVM, to setup machines and make them available to TORQUE on the fly. TORQUE, on the other hand, is not aware of any setup behind the scenes, instead, it schedules jobs in the normal way so all its advanced scheduling features are retained. ViBatch consists of a set of scripts: it uses TORQUE’s Prologue and Epilogue scripts to
start and stop virtual machines; and a remote shell script to run the actual job in the assigned VM. A user normally submits jobs to a specified queue and a VM is then created based on a template with an image. After the VM is started an SSH connection is created to copy job scripts and other data to the VM and the job’s execution shell is directed to this VM to run the job. The VM will be destroyed after the job finishes execution. ViBatch doesn’t work in a cloud environment.

3. Dynamic TORQUE, integrating TORQUE/Maui and OpenStack

When running TORQUE with the Maui job scheduler there is no built-in mechanisms to interact with an OpenStack cloud. An approach to overcome this limitation is to develop an external component that runs alongside TORQUE and Maui and communicates with the OpenStack API to control the creation and deletion of VMs. This external approach has two major benefits; firstly it is transparent to users since the users continue to submit jobs to the TORQUE queue in the same way and don’t need to know anything about the cloud, and secondly if the external service crashes TORQUE will not be affected and can continue to process jobs.

Dynamic TORQUE is a tool we’ve developed based on this integration approach and provides the bridge between a TORQUE/Maui batch system and an OpenStack cloud service. It can be configured to use either an active or a passive mode of communication with the TORQUE/Maui batch system. A description of each mode followed by a comparison of each approach is now presented.

3.1. Active mode

![Dynamic Torque running in Active Mode](image)

Dynamic TORQUE’s active mode works by actively querying the TORQUE/Maui job queue at fixed intervals. If all existing worker nodes are busy but there are still jobs waiting Dynamic TORQUE will launch new VMs in the cloud based on the job priorities gathered from Maui and the resource requirements of these jobs. Dynamic TORQUE then queries the states of the newly-launched VMs to determine if they are ready or not. Typically it takes around 5 minutes for the OpenStack cloud to provision a VM and extra time to execute a server customization script. When a VM is ready Dynamic TORQUE adds it to the TORQUE server as a normal worker node. TORQUE/Maui can then distribute idle jobs to the new worker node.

Dynamic TORQUE also provides VM worker node monitoring by querying VMs to see if any node has been idle for too long or is nonresponsive. If one of these two states is detected Dynamic TORQUE will remove it from TORQUE’s worker node list and wait for Maui to update its
information about this node. This ensures the consistency between TORQUE and Maui, otherwise when the VM is removed from TORQUE, Maui still thinks it is there. Then it is safe to send a request to shut down this VM in the cloud. The deletion routine checks its status in the cloud periodically until it has disappeared from the cloud, this process makes sure no residue VMs are left in the cloud in an abnormal state.

As VMs are added to TORQUE as normal worker nodes, they need to be set up like other physical worker nodes. Firstly, a PBS MOM process needs to run on the VM in order to communicate with the TORQUE server. Secondly, the VM needs to know all user information of the system; generally it should be configured with a central LDAP server. Lastly, the VM needs to be able to access a shared file system, so the jobs run on this VM can see the input data and write output data. These can be built into the image that is used to launch the VM, or done in the VM customization step. In active mode it is similar in approach to how Cloud Scheduler allows the Condor batch system to make use of cloud services.

3.2. Passive mode
In passive mode Dynamic TORQUE is integrated with a special TORQUE worker node. In the context of TORQUE all the resources in the cloud are like a big worker node with many cores. The Dynamic TORQUE service that runs on the special worker node is called VM Pool and it has the configuration details of the cloud that it interacts with. VM Pool maintains a pool of VMs in the cloud that are used to run jobs. The special TORQUE worker node uses prologue and epilogue scripts to interact with VM Pool. VM Pool stays silent unless a job comes into TORQUE and is distributed to the special worker node. The workflow is illustrated in Figure 2 and described below:
1. The user submits a job to the TORQUE server.
2. The TORQUE server sends the job to the special worker node configured with the VM Pool prologue and epilogue scripts.
3. The Job starts if there is enough resource (in TORQUE’s config, the worker node is configured to have X number of cores where X is the maximum number of cores that can get from the cloud), which means not all cloud resources are being used
4. The TORQUE MOM daemon (on the special worker node) runs the prologue script that requests a VM from the VM Pool. On success, the script will initiate the VM so that the VM is able to run the given job. This initialization is specific to the job, while a more generic VM initialization can be performed when the VM is launched. The job-specific initialization includes the creation of the user account that will be used to run the job, the creation of any directories for the job, and copying the job description file from MOM to the VM in the cloud. If all VMs managed by VM Pool are busy, prologue will not be able to get a free VM, thus it returns an error code to the TORQUE MOM which will put the job back in the TORQUE queue. After a period of time the job will be tried again.
5. The TORQUE MOM executes the job using Remoteshell script. When submitting the job the user needs to specify to use a special Remoteshell script. This script firstly gets the IP of the assigned VM from the VM Pool, then creates an SSH connection to the VM and runs the job description file on the remote VM.
6. Once the job is finished the epilogue script returns the VM back to VM Pool and cleans up other job related files.

The VM Pool and TORQUE MOM do not rely on each other. If VM Pool stops running, the TORQUE MOM will be marked as being down by the automatic node health check script. Thus no jobs will be sent to this worker node. The VM Pool maintains the lifecycle of its VMs, from spawning to destruction. If a VM is not usable for any reason it will be shut down and replaced by a new one.

Dynamic TORQUE’s passive mode is inspired by ViBatch and has used similar techniques for cloud integration.
3.3. Comparison

Both of Dynamic TORQUE’s operating modes have their merits and disadvantages. The comparison below and knowledge of a batch system’s anticipated workload will help to determine which mode will behave best with a particular set of circumstances.

Active mode takes most advantage of TORQUE’s existing features since in essence it is simply adding normal worker nodes to TORQUE. This includes Maui’s scheduling policies which are adhered to when running jobs, and worker nodes are shared by jobs. Maui does occasionally get confused when worker nodes are added to or removed from a running TORQUE queue. A Maui restart is required if it is inconsistent with the TORQUE server on such occasions. Jobs are distributed by TORQUE’s MOM, so from a user’s perspective, they are submitted jobs to the usual queue, without any changes.

On the other hand, passive mode does not change any configuration in TORQUE or Maui; it only uses TORQUE’s built-in mechanisms, with a minor change in qsub command. However, as all cores are configured into one worker node, MAUI policies are not much useful in resource allocation; there needs some external routine in VM Pool to better schedule jobs. VM Pool requires a scheduler to decide the flavor of the VM to launch, the flavor of an existing VM to shut down. In our implementation, we made an assumption that all jobs need the same number of cores, and all VMs have the same number of cores as the jobs; therefore one VM is dedicated to run one job. A VM cannot be shared by several jobs like active mode. Table 1 gives a summary of the comparison.

| TORQUE integration          | Active mode | Passive mode |
|-----------------------------|-------------|--------------|
| Job distribution            | Run on TORQUE server (head node) | Run on TORQUE worker node |
| Worker node allocation      | TORQUE/Maui | External scripts |
| Worker node usage           | shared      | VM Pool |
| Scheduling policy           | TORQUE/Maui | TORQUE/Maui + VM Pool |
| User experience             | Transparent | Modified qsub |
| Queues to watch             | Multiple    | Multiple     |

In our test system, we are using active mode because we want to take advantage of Maui’s scheduling policy and make it transparent to users. Details will be given in the next section.
4. A Tier-3 use case

The CoEPP Tier3 (a Tier 3 compute cluster using the NeCTAR cloud) has TORQUE running with dynamic provisioning capability in active mode. The TORQUE server (head node) is running in a CoEPP-hosted physical machine off the cloud, along with other key components, as shown in Figure 3. Only worker nodes and submit nodes are running in the cloud. The reason is that CoEPP has human and hardware resources to host key components, which are unlikely to change and need to be stable all the time. The cloud resources are used to run dynamic components that can be shut down any time. But technically everything could be run in the cloud.

![Figure 3. CoEPP’s tier 3 system](image)

Dynamic TORQUE is the external service running on the head node, as described in section 3.1. It orchestrates worker nodes in the cloud via the OpenStack APIs. The worker nodes in the cloud are grouped as ‘static’ and ‘dynamic’: static means these worker nodes stay up forever, no matter what the current workload is; dynamic means the worker nodes will be shut down if no job is in the queue, and they will be launched if jobs are waiting in the queue. The numbers of static and dynamic nodes are configurable and can be changed any time. Dynamic TORQUE also monitors the health of worker nodes. If a static worker node is down and inaccessible, it will shut it down and fire up a new one to replace it. The same Torque can also have ‘physical’ worker nodes that are running off the cloud, in which case Dynamic TORQUE supports cloud bursting from physical static cluster to dynamic virtual cluster in the cloud.

All worker nodes are launched from one single VM image, which makes it easy to maintain. This image contains basic configurations so that the VM can work in this environment once it is booted up: NFS is used as the shared file system and mounted on all worker nodes; a central LDAP is deployed to provide user information to all worker nodes. A Puppet\(^1\) agent is also running in the worker node so that it can pick up any new changes if needed. We also set up a monitoring system, including Ganglia\(^2\) and Nagios\(^3\), to monitor the usage of the system. In the cloud there are also a few submit nodes, which are for users to program and then submit jobs to TORQUE. They are not part of the Dynamic TORQUE territory but we have developed other routines to easily add or subtract any number of submit nodes according to users’ needs.

This system is currently in production and serving Tier 3 users. Dynamic TORQUE sends data to Ganglia for usage tracking. Figure 4 shows the Ganglia graphs about Dynamic TORQUE in a period of one day: the graph on the left shows the usage of cores; the graph on the right shows the number of queued and running jobs. Note the increase in the cluster size near 06:00 as Dynamic Torque detected

---

\(^1\) [http://puppetlabs.com/](http://puppetlabs.com/)

\(^2\) [http://ganglia.sourceforge.net/](http://ganglia.sourceforge.net/)

\(^3\) [http://www.nagios.org/](http://www.nagios.org/)
increased demand. The additional cores created were subsequently released after 20:00 as demand dropped.

![Figure 4. Dynamic TORQUE in operation as monitored by Ganglia](image)

5. Conclusion and future work

This paper describes how to run TORQUE/Maui with dynamically allocated resources in the cloud. TORQUE was originally designed for a compute cluster in a static environment. It is a challenge to run it in the cloud because we don’t want to keep worker nodes up unless they are required. There needs to be a capability to expand and shrink worker nodes in the cloud according to current workload. Some work has been done by others to run queueing systems in the cloud, however, there is no solution suitable for TORQUE/Maui.

We have investigated different approaches and tested them in our cloud environment. We then developed a solution and delivered it to our users. The chosen solution involves running an external service that queries TORQUE and Maui at fixed intervals then, based on the job priorities, launches necessary worker nodes in the cloud and hands them over to TORQUE. Thus TORQUE/Maui can use its existing scheduling policies to make good use of these resources. Once the workload decreases idle resources will be removed from TORQUE and returned back to the cloud. This approach has proven to be success for our users.

The NeCTAR research cloud consists of several cells, or availability zones. Currently Dynamic TORQUE can only launch worker nodes in one cell because all data is in one place. In the future our users will be able to acquire storage resources in other cells and place data in multiple locations. Thus Dynamic TORQUE will need the ability to create VMs in different cells allowing jobs to be assigned to VMs that are close to the data.

References

[1] NeCTAR. [http://nectar.org.au/research-cloud](http://nectar.org.au/research-cloud) (viewed October 2013)

[2] TORQUE. [http://www.adaptivecomputing.com/products/open-source/torque/](http://www.adaptivecomputing.com/products/open-source/torque/) (viewed October 2013)

[3] Maui. [http://www.adaptivecomputing.com/products/open-source/maui/](http://www.adaptivecomputing.com/products/open-source/maui/) (viewed October 2013)

[4] DIRAC: [http://diracgrid.org/](http://diracgrid.org/) (viewed October 2013)

[5] R. G. Diaz, A. C. Ramo, A. C. Agüero, T. Fifield and M.E. Sevior Belle-DIRAC 2011 Setup for Using Amazon Elastic Compute Cloud Journal of Grid Computing Volume 9, Number 1, 65-79

[6] P. Armstrong et al. 2010. Cloud Scheduler: a resource manager for distributed compute clouds. CoRR, abs/1007.0050

[7] Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[8] StarCluster. [http://star.mit.edu/cluster/](http://star.mit.edu/cluster/) (viewed October 2013)

[9] ViBatch. [https://ekptrac.physik.uni-karlsruhe.de/trac/BatchVirt/wiki/ViBatch](https://ekptrac.physik.uni-karlsruhe.de/trac/BatchVirt/wiki/ViBatch) (viewed October 2013)