Advancing Research Infrastructure Using OpenStack

Ibad Kureshi, Carl Pulley, John Brennan, Violeta Holmes, Stephen Bonner, Yvonne James
School of Computing and Engineering
University of Huddersfield
Huddersfield, UK HD1 3DH
Email: hpc-rg@hud.ac.uk

***Abstract*—Cloud computing, which evolved from grid computing, virtualisation and automation, has a potential to deliver a variety of services to the end user via the Internet. Using the Web to deliver Infrastructure, Software and Platform as a Service (SaaS/PaaS) has benefits of reducing the cost of investment in internal resources of an organisation. It also provides greater flexibility and scalability in the utilisation of the resources. There are different cloud deployment models - public, private, community and hybrid clouds. This paper presents the results of research and development work in deploying a private cloud using OpenStack at the University of Huddersfield, UK, integrated into the University campus Grid QGG.**

The aim of our research is to use a private cloud to improve the High Performance Computing (HPC) research infrastructure. This will lead to a flexible and scalable resource for research, teaching and assessment. As a result of our work we have deployed private QGG-cloud and devised a decision matrix and mechanisms required to expand HPC clusters into the cloud maximising the resource utilisation efficiency of the cloud.

As part of teaching and assessment of computing courses an Automated Formative Assessment (AFA) system was implemented in the QGG-Cloud. The system utilises the cloud's flexibility and scalability to assign and reconfigure required resources for different tasks in the AFA. Furthermore, the throughput characteristics of assessment workflows were investigated and analysed so that the requirements for cloud-based provisioning can be adequately made.

I. Introduction

Since 2010, the University of Huddersfield has established a private cloud. The primary aims for this system is to further the research goals in advanced computer systems. The cloud was quickly adopted by other academics and this system went from being a research machine to an integral part of the campus grid infrastructure. As a whole, this infrastructure is known as the Queensgate Grid (QGG), and includes: several high performance computing (HPC) clusters; a render farm; and a high throughput (HTC) cycle stealing system. Now this private cloud provides not only Infrastructure as a Service (IaaS) but platform and software services as well.

Heavily used in delivering courses like 'Network Security' and 'Operating Systems', the QGG-Cloud has served as a 'Virtual Laboratory'. With minimal hardware and very little technician time, the students were provided with a learning environment to meet their academic needs. Different flavours of the Microsoft Windows®and Linux operating systems were delivered over IP. This has meant that: the technicians do not have to exhaust time dual/triple booting lab machines; students can be given access to many different operating systems; and with more enhanced access privileges than if they were using a physical machine plugged into the University network. Previously the specialist labs that could dual/triple boot were not available out of hours, due to their physical locations, but using the QGG-Cloud students can get access to their learning environment from the 24 hours labs within the library [1].

To help researchers and students undertaking research projects, different applications are placed in the cloud as snapshots of typical lab PCs. This system of creating snapshots also ensures backward compatibility of software. Even if new versions of the software or operating systems are released, researchers can still recreate previous conditions when proofing their prior research. Delivering Software as a Service in this manner allows the users to get different (in most cases higher) hardware specifications to do their science. Provisioning faster CPU's or larger memory configurations ensures the University and research capital is not wasted on an endless cycle of purchases. For most of their lifetime the cutting edge desktop machines are either idle, not being used for their intended purpose or not being used to their maximum capability [2]. The approach of centrally providing high memory type configurations leads to better access for all users within the institution, improved utilisation and better management of power hungry devices.

The QGG-Cloud in its current incarnation is an Open Stack Grizzly deployment using RDO over CentOS 6.4 [3]. The central head or access node provisions images on quad socket AMD based servers. These servers have 4GB of RAM per core. The system can provide up to 96 standard m2.medium configurations. Internally the cloud has two gigabit networks and externally is linked to the University's 10 gigabit research computing network.

In this paper we expand on work described in the paper 'Using OpenStack to Improve Student Experience in an H.E. Environment'[1]. In the previous work we briefly described how the private IaaS cloud supported our existing research computing infrastructure by providing machine configurations not available in the traditional HPC clusters. In Section II, of this paper, we describe how the challenges of dynamically surging our traditional HPC workloads to the cloud have been met. The ability to dynamically scale beyond the rigid hardware of the clusters was achieved by making enhancements to the open-source HPC batch system TORQUE and the job scheduler MAUI.

The growing numbers of students enrolling for University and online programming courses requires the use of automated evaluation of student assignments in order to meet that demand. Previous work[1] demonstrated a system (based on a custom Domain Specific Languages (DSL)) for specifying

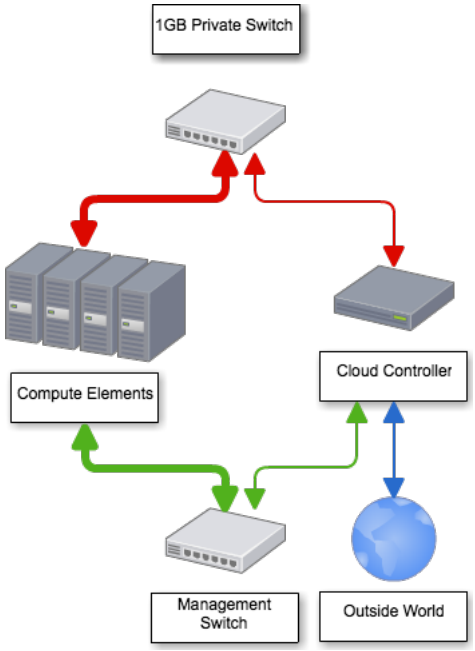

Fig. 1: Layout of the OpenStack based QGG-Cloud

Automated Feedback Assessments (AFA) and was delivered to academics using a Platform as a Service. We expand on that work here by examining an Akka actor[4] implementation of an assessment workflow. In Section III we use actual student submissions[1] along with the original submission frequency model, to simulate different loading scenarios for our assessment workflows. In using an actor based implementation, we additionally investigate the extent to which claims of scaling and failure resilience hold.

II. Surge Computing

A. Background

Cloud computing has been described as the next paradigm in computing. It promises to deliver computing power as the fourth utility direct to the end user. While large enterprises have yet to move their IT infrastructure into the cloud, startups and small businesses have benefited greatly from the cloud. The shorter barrier to entry means that companies can deliver their product or enhance their infrastructure as needed [5].

Where small to medium enterprises (SME) have successfully employed cloud computing is in scaling their existing infrastructure, as and when required. Whilst they are still reliant on some internal resources, in moments of heavy load the SME can scale their resources within a public cloud. The companies can therefore guarantee customers a high quality of service with minimal investment. This form of provisioning is known as surge computing [6].

Most current work is focused on surging Web 2.0 type workloads within the cloud [7][8][9]. Using load balancing and heartbeat monitoring tools, Apache web servers and mySQL databases are provisioned to meet such high demands.

[1]Our corpus of student data consists of 2164 submissions made by 163 students over a 210 day period.

There are many proponents of cloud computing whom additionally advocate migrating high performance computing workloads to the cloud [10][11][12]. However, there has been little existing work in creating a HPC Job Manager and Scheduler that is truly dynamic and elastic. The HTCondor project is a dynamic HPC scheduler, but it lacks elasticity, working on the principle that execution nodes can start up and then connect to the control node. The control node itself does not 'hunt' for execution endpoints. Even with such dynamism HTCondor can be utilised for HPC type workloads. It has been utilised by pharmaceutical companies to create large clusters on Amazon EC2 [13]. The major limit to the dynamism currently provided is that based on the workload on the head node there is no mechanism to automatically generate execution endpoints.

Traditional HPC Job managers are very rigid. Job managers like TORQUE and Grid Engine need to be given endpoint information, e.g. hostnames or IP addresses, at startup. If a change is required then the whole job management suite needs to be restarted. This obviously makes such schedulers inelastic. Recently however, IBM (in collaboration with Platform Computing) have released a version of the Platform LSF that dynamically creates nodes within elastic environments to meet such HPC needs. The Enterprise Edition of the MOAB HPC Suite now also includes dynamic cloud based provisioning functionality. Penguin Computing too have a cloud based initiative for enterprise clients [14].

Whereas the previous two solutions are proprietary, our efforts to surge enable the local HPC have solely utilised open source GNU/GPL job management systems.

B. Motivation

The University of Huddersfield's research computing grid comprises of several high performance compute clusters. They differ from each other by core speed, interconnect speed, co-processor availability and memory configurations. On average most systems have a 2GB RAM per core core memory configuration. The maximum available memory configuration is 4GB RAM per core, but these nodes cap off at 16GB RAM per node [15]. Researchers at Huddersfield do have access to Shared Memory Processor (SMP) and POWER systems based high memory configurations through academic partnerships with other institutions. Unfortunately, there are some applications within the QGG that are not able to run on non x86 systems. Others have license limitations that prevent the software from being used off-campus. Additionally, there are instances where researcher generated models can not be partitioned to run on a distributed memory clusters [16].

An example scenario has occurred with researchers from the school of Human and Health Sciences (HHS). Using a commercial *NIX based rendering package, the researchers generate high resolution visualisations of blood flow. Under most conditions the rendering package divides the frames between nodes and speeds up the render time. However, for the HHS renders, each single frame can consume more than the maximum available 16GB of RAM. Due to the commercial nature of the software package, the application can not be ported to other architectures, nor can it be deployed on a partner institutions HPC system. Before HPC and then cloud deployment of this package, HHS bought 8GB RAM

workstations, followed by upgrades to 16GB and then finally a single 32GB RAM workstation. The purchasing power of the research group has quite clearly been affected by a single computational problem.

The purchased workstations are power hungry in an idle state and, for the majority of the time, are used for standard office work! Provisioning for such applications within the QGG-Cloud allows for delivering different hardware configurations that the researcher could not possibly get under their desk. The key however is to remove the complexity of the cloud and to seamlessly integrate this system within the existing HPC infrastructure so that the researchers workflow is not significantly disrupted.

C. Implementation

In the initial development, we designed a wrapper script for PBS/TORQUE (TORQUE being the commonly deployed batch system within the QGG). Using Python extensions for Keystone, and a MySQL database to hold configurations, the wrapper script decides if a job should be directly submitted to the underlying torque system or surged to the QGG-Cloud.

In section II-C1 we outline the decision making process undertaken before jobs are surged to the cloud. In section II-C2 we provide detailed explanations for the mechanisms required to expand the TORQUE cluster into the cloud.

1) Decision Metrics: When a job is submitted to the cluster, the wrapper intercepts the job file. If the requested resources (in the job file) match those resources that can be found within the hardware scope (of the traditional cluster), then the job carries on through the system as normal. If the requested resources exceed what is available as bare metal, then the wrapper begins to provision a node in the cloud. Using information defined in the configuration database, along with the status of the load on the cloud, decisions are taken regarding the provisioning. A flow chart showing the steps is shown in Figure 2.

If the requested resources can fit on pre-defined cloud limits, then the wrapper initiates a compute node instance within the cloud and submits the job to TORQUE, along with a flag that ensures that this job will be routed to the new cloud instance. If the requested resources exceed the limits of the local cloud service, and the user/administrator has configured credentials to surge to a public cloud, then the VM is instantiated within the public cloud. In the event that there are no public cloud credentials, the job is returned to the user with an error message.

To maximise the utilisation efficiency of the private cloud, the wrapper is able to create a hardware flavour within cloud. This flexibility is essential in a private cloud setting. Generally provision options on clouds tend to be configured in binary increments. So after a 16GB RAM flavour the next flavour will have 32GB of RAM. Therefore if a user requests 20GB of RAM for their simulation there will be 12GB of RAM booked and not used. It will also limit the total number of surge instances created.

The following section covers the actual methodology of implementation for the wrapper.

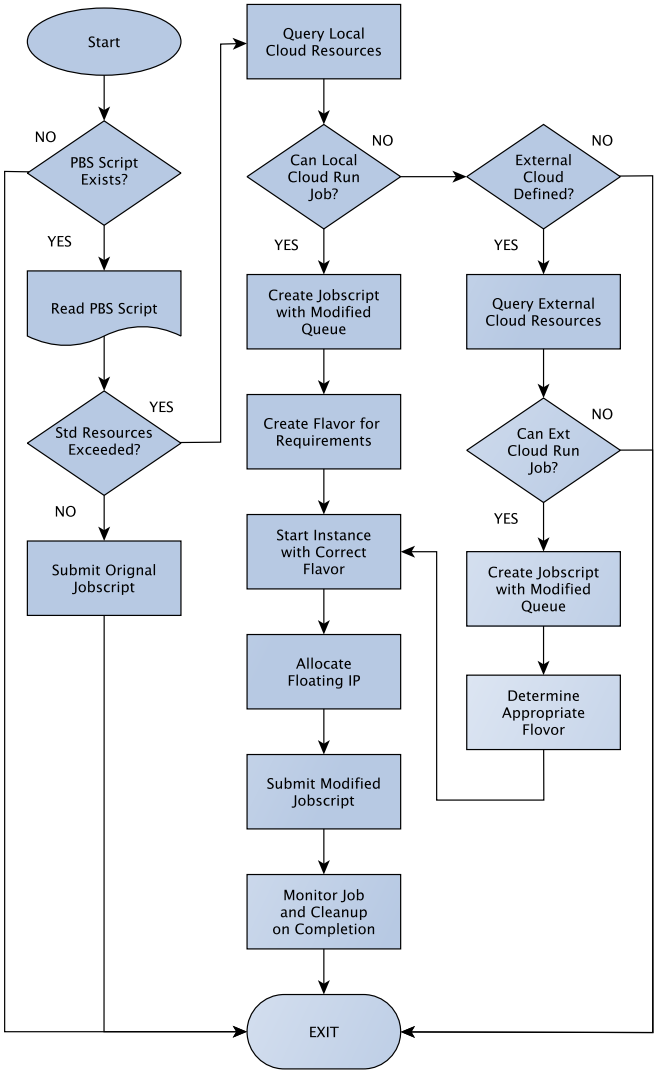

Fig. 2: Flowchart depicting the decision making process in HPC Cloud Surging

2) Surge Wrapper for TORQUE: Users on the surge enabled HPC system have a custom script named 'qsub' in their default paths. This qsub is not the standard TORQUE submission executable. The script passes the users submission to the surge wrapper. The surge wrapper parses the arguments supplied by the user to PBS/TORQUE. If a job is submitted with more than the job file argument, the wrapper automatically passes the job down to the scheduler. To invoke the surge a user has to define their job within a PBS job file.

Using PBS based functions, the wrapper queries the underlying HPC system to assess the available capabilities. Once it has ascertained that the requested resources are not natively available via TORQUE, it calls the function to surge the job into the cloud. Using the limits specified by the configuration database, the wrapper then generates a hardware flavour template. This is injected into OpenStack. The configuration database holds information such as: how elastic the system is; the largest flavour instance possible; etc.

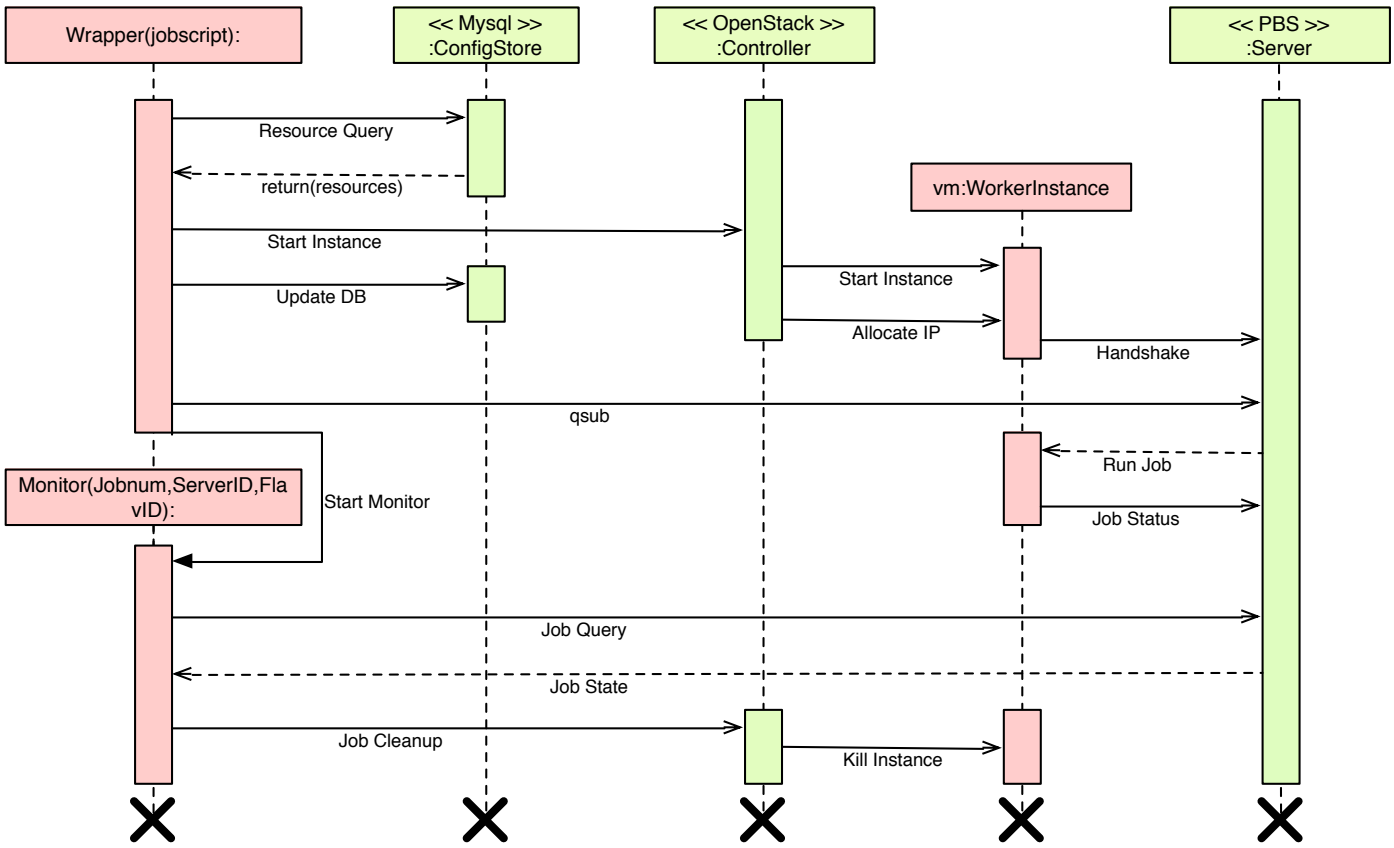

Fig. 3: Control bus depicting the sequence of events in an HPC Surge

With the hardware flavour injected into the system, the wrapper starts a virtual machine instance utilising a pre-configured image. This image mirrors a typical node within the HPC system. Next, the job script is regenerated with a special queue name and handed over to TORQUE. The wrapper also starts a small service to monitor the job's progress.

On the system TORQUE is pre-configured to recognise 'n' number of nodes (where n is equal to the maximum nodes that can be generated in the cloud). These nodes are locked to the special queue. So when the wrapper submits the job with the modified queue, TORQUE waits for the instance in the cloud to start up and then matches the job to the cloud instance. From the TORQUE point of view, a job has arrived which can only be assigned to a particular set of nodes and, one of those nodes has just become available.

Within the cloud, OpenStack generates a new hardware flavour and instantiates an image against this flavour. It then subtracts the requested resources from the pool of resources the surge can request. This is fed back to the surge wrapper to ensure the cloud does not get overly subscribed.

Upon completion of the job, the TORQUE MOM demon becomes idle and, once the job monitoring service detects this, injects a poison pill to terminate the VM instance. The custom hardware flavour is additionally removed from the OpenStack environment. Within TORQUE, the node becomes offline. Figure 3 depicts the sequence of events that take place when a job is surged to the private cloud.

Whether being used in a public or private cloud environment, the systems scalability will always be controlled. In a public cloud the availability of resources purchased (or some form of financial cap) will prevent an indiscriminate number of nodes being spawned. In a private cloud the elasticity itself is limited. OpenStack controls the number of cores and instances a user can run through group policies. In OpenStack parlance, this is know as 'Project Quotas'. In the situation where the surge 'project' on OpenStack is out of resources to spawn a new node, the new job is simply queued. If the new job, e.g. job-B, can run on already spawned virtual machines, then job-B is sent to TORQUE in the modified queue. The monitoring service that is awaiting the completion of already started/running jobs, is passed the new jobs information. Upon completion of the running jobs, the monitor refrains from injecting the poison pill, allowing TORQUE to push job-B to the now idle node. If the spawned instances do not meet the requirements of the new job, then job-B is held in a queue external to TORQUE. Running jobs will terminate as normal, and instances are brought down returning their resources to the quota. The job is then reconsidered for execution when the monitor reports a change to the load on the cloud.

If a requested hardware configuration can never be met, an error is returned to the user. Information pertaining to the maximum number of nodes, smallest configuration of nodes, largest configuration possible, static flavours (if using a public cloud) and credentials, are available to the wrapper from the MySQL configurations database.

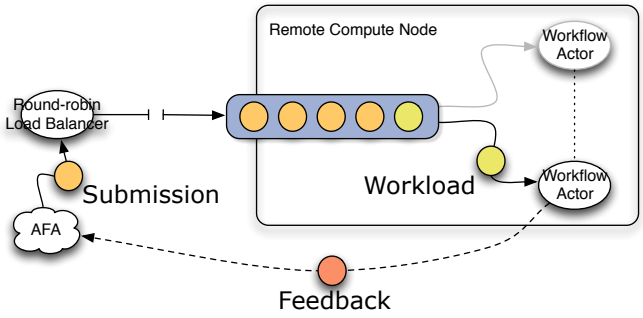

Fig. 4: Assessment Workflow

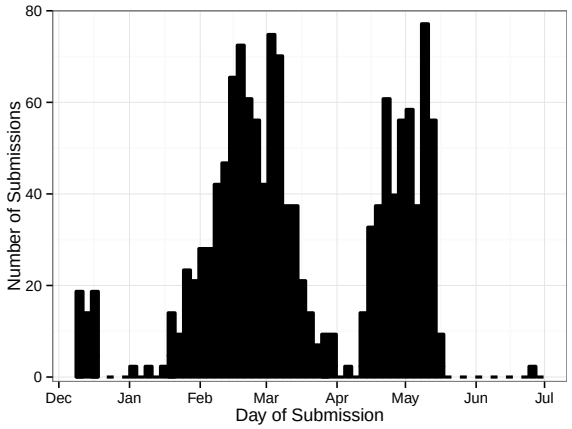

Fig. 5: Student Coursework Submission Frequency

The HPC Surge completed our institutional demands for Infrastructure as a Service. In the next section we evaluate the Automated Feedback Assessment framework, which is delivered to our users as Platform as a Service.

III. AFA Workflows

In [1] we investigated how the HPC cloud could be used to automatically generate formative feedback for submitted student assessments [2]. Using a combination of various technology stacks (i.e. JClouds, Chef, Apache Camel and Akka actors), [1] presented a domain specific language (DSL) that:

- could define complex assessment workflows
- was able to support a (extensible) variety of data collection, analysis and feedback delivery methods
- and simplified the deployment and configuration of the underlying computing infrastructure.

In this paper, we focus on the assessment workflows that AFAs encapsulate and investigate their design for throughput and scalability. Our assessment workflows are profiled using a corpus of student work[3] taken from a real-world (topological sorting) Java programming exercise.

A. Workflow Design

Assessment workflows are encapsulated by an AFA that is responsible for managing the consuming of coursework submissions, their injection into the assessment workflow and the subsequent processing and delivery of the (workflow returned) submission feedback[1]. As assessment workflows are the resource intensive component of an AFA, here we propose resilient and scalable designs for them.

[17] investigates how a combination of testing, control-flow graph similarity matching and software verification can be used to effectively assess and grade student code. Our work completes this work by providing a framework within which their assessment strategy, or workflow, may be run. Unlike [17], we base our work on a more complex programming exercise, and so face additional challenges in using software verification. Future work plans to explore the extent to which we may address these issues using Separation Logic based solutions such as: jStar[18] and Krakatoa[19].

Figure 4 shows our assessment workflow deployment strategy. Using a round-robin load balancing router, we send messages (i.e. submissions) to a series of remote cloud compute nodes. On each cloud compute node, messages are received and processed by a collection of worker actors that all share and process the same mailbox (configured via a balancing dispatcher). After a worker actor has processed a message, it replies with its result (i.e. feedback) to the message's original sender.

On going work is investigating how best to implement message persistence when cloud compute nodes fail or restart. Currently, we have used durable mailboxes (implemented using filesystems) to provide resilience against restarts. Future work will investigate using event sourcing[20] to rebuild system state by replaying submission events from message transaction logs.

B. Methodology

Our aim is to investigate the throughput characteristics of realistic assessment workflows so that requirements for cloud based provisioning can be adequately made. We first describe a workflow model and then how we plan to load it with submission data.

As reported in [17], program based assessments consist of three workload tasks: testing; control-flow graph similarity matching; and software verification. The resource requirements for each of these three tasks are dependent upon the specific submission and their processing may occur concurrently. Hence, we approximate our workflow model by simply performing testing only in our simulations. For our simulations, we configure our experiments to use a single (local) compute node, with 8 worker balanced workflows.

In order to measure quality of service, we constrain queued submissions so that they must be processed within a given time period (we choose a value of 5 minutes). Any feedback replies generated after the timeout period will be rejected. We implement this timeout using an Akka actor that proxies

[2] All code is open sourced and publicly available from: http://github.com/carlpulley/cloud-paper.

[3] The corpus consists of 2164 submissions made by 163 students over the period of 201 days.

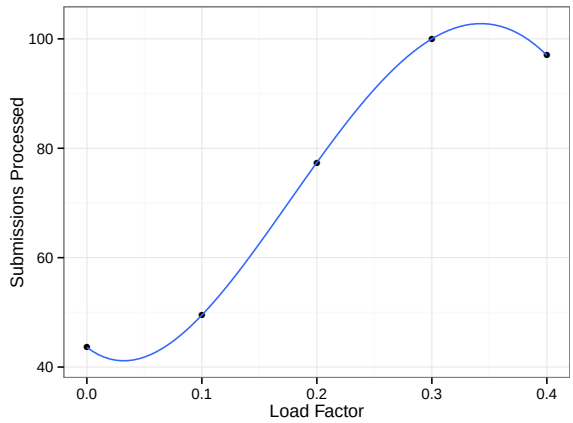

Fig. 6: Load Factor vs Submissions Processed

Load Factor	Submissions Processed (%)	Avg. Processing Time (s)
0.4	97.0	0.42
0.3	100.0	0.42
0.2	77.4	0.42
0.1	49.5	0.40
0.0	43.6	0.42

TABLE I: Resource Usage Data

the workflow actor using the implicit timeout of an *ask* or request/reply message pattern.

Different delivery loads are simulated using the actual submission frequency (see figure 5) of our real-world student data scaled with a constant *load* factor.

For each *load* factor, profiling experiments were ran for a fixed time of 15 minutes, with data collected at the termination of that experiment. All simulation code was ran on a 2.4GHz Quad-core Intel Xeon Mac Pro with 26GB of RAM, using a JVM configured with 256MB of stack and 1GB of heap space. Resource monitoring was performed using the Typesafe console. All other Akka actor configuration was set to the default values.

C. Data Analysis

By examining the AFA's message store, we extract the time of first submission, the last accepted feedback and the number of submissions and accepted feedbacks. From this data we can calculate (see table I) the percentage of submissions processed (our throughput) and a submissions average processing time (a quality of service measure).

With a load factor of 0.4, only 97.0% of submissions were processed. This is an artefact of the time period over which the experiments were run. If they had been run for a longer period (e.g. 20 minutes), then existing feedback processing would have completed and the expected 100% throughput for this load factor would have been achieved.

Throughout, our average processing time remains reasonably constant. Moreover, as we flood the actor queues with submission messages (i.e. the load factor gets smaller), the number of successful feedbacks generated decreases. This is expected behaviour. Using a graph plot of load factor vs submissions processed (see figure 6), along with an estimate of acceptable feedback loss and compute node startup time, we can now predict the point at which more compute nodes should be launched. By monitoring our feedback message failure rates, our Akka actor workflow can be configured to elastically adapt to the changing workflow demands.

IV. Further Work

One of the biggest shortfalls with present job schedulers is their rigidity. The TORQUE batch queuing system needs to know about all execution endpoints and their capabilities before accepting a job. If a user requests more hardware than can be provided, the job is not accepted. This is why our implementation required a wrapper to sit above TORQUE assessing the requirements of incoming jobs and then provisioning for the demand. Our aim is to integrate the wrapper into a hybridised version of TORQUE. In this new elastic TORQUE, a job will not be rejected because of a lack of resources. Using the same principals of the surge wrapper, TORQUE will assess if it can initiate a node in the cloud to provision the required hardware. A job will only be rejected if the job's requirements can truly not be met.

The current implementation of the surge tool lacks interoperability with other cloud providers and middlewares. The final version of the wrapper tool will be more modular and have provisions to plug in to different cloud middlewares and providers. While current public clouds maybe too costly for large scale HPC provisioning, the public cloud can be used to surge time critical workloads to, or to get access to hardware configurations (such as GPU etc.) that are not available in house.

Within the University of Huddersfield's research computing grid, there is currently research taking place to make traditional HPC schedulers more intelligent [21]. Using an open-framework bench-marking suite known as the 'Application and System Performance Profiler', job schedulers are being given more information and are thus more aware of the performance characteristics of the software being executed on the HPC clusters. Heuristic information, gathered from previous jobs, is also being utilised in the scheduling decisions. Using the dynamic scaling features that the elastic-TORQUE project provides, the Intelligent scheduler can lead to cloud environments being utilised more efficiently for HPC workloads. Performance profiles will provide the HPC job manager with estimated resource requirements and the corresponding run times, so that the scheduler can factor in a new dimension to its scheduling algorithm: cost. This will truly open up the cloud for HPC and other research computing enterprises.

V. Summary and Conclusion

Cloud computing has great potential to provide scalable and flexible computational resources in commercial and academic environments. However, concerns regarding intellectual property rights and security risks, combined with the high cost of acquiring resources through a commercial provision have stimulated research into private clouds for education and research institutes.

In this paper we have presented the results of research in deploying and utilising cloud technology to create the QGG-Cloud at the University of Huddersfield, UK. This private

cloud is deployed as an OpenStack Grizzly using RDO over CentOS 6.4. It supports users needs for High Performance Computing resources by providing machine configurations not available in traditional HPC clusters, delivered as IaaS. To improve resource utilisation within the cloud we have implemented an HPC job manager and scheduler that are dynamic and elastic. This was achieved by making an enhancement to the open-source HPC system TORQUE and the job scheduler MAUI. Future work will focus on developing an intelligent scheduler to further improve utilisation of cloud environments for HPC workloads.

Responding to the demand for quicker and more insightful coursework feedback from computing students, we have implemented Automated Feedback Assessment (AFA) system within the university private cloud, delivered as PaaS. Based on the analysis of student coursework submission frequency, we have devised assessment workflow deployment strategy and investigated the throughput characteristics of the workflows. This has allowed workflows to be configured to elastically adapt to changing demands, and predict when more computing nodes should be launched in the cloud.

Building on the University's existing investment in internal HPC clusters and campus grid technology, and using open source cloud software - OpenStack, we have deployed the QGG-Cloud. This private cloud provides an effective solution for advancing the HPC research infrastructure and a flexible and scalable system used for research, teaching and assessment at the university.

ACKNOWLEDGMENT

The authors would like to acknowledge the use of the University of Huddersfield computational grid the Queensgate Grid.

REFERENCES

[1] Bonner, Stephen, Pulley, Carl, Kureshi, Ibad, Holmes, Violeta, Brennan, John and James, Yvonne (2013). *Using OpenStack To Improve Student Experience in an H.E. Environment.*, Science and Information Conference 2013, London, pp.888-894. TheSAI

[2] Sundararajan, E.; Tan Bing Yean; Rahmani, M., *"Using computer labs for distributed computing,"*, 2011 International Conference on Electrical Engineering and Informatics (ICEEI), pp.1,6, 17-19 July 2011

[3] "Redhat Open Stack" RDO. Redhat, n.d. Web. http://openstack.redhat.org/.

[4] Wyatt, Derek (2013). *Akka Concurrency.* Artima Developer.

[5] Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008, November). *Cloud computing and grid computing 360-degree compared*. In Grid Computing Environments Workshop, 2008. GCE'08 (pp. 1-10). IEEE.

[6] Van den Bossche, R., Vanmechelen, K., & Broeckhove, J. (2010, July). *Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads*. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on pp. 228-235. IEEE.

[7] Chieu, T. C., Mohindra, A., Karve, A. A., & Segal, A. (2009, October). *Dynamic scaling of web applications in a virtualized cloud computing environment*. In IEEE International Conference on e-Business Engineering, 2009. ICEBE'09. pp. 281-286. IEEE.

[8] Vaquero, L. M., Rodero-Merino, L., & Buyya, R. (2011). *Dynamically scaling applications in the cloud*. ACM SIGCOMM Computer Communication Review, 41(1), 45-52.

[9] Mao, M., Li, J., & Humphrey, M. (2010, October). *Cloud auto-scaling with deadline and budget constraints*. In International Conference on Grid Computing (GRID), 2010 11th IEEE/ACM. pp. 41-48. IEEE.

[10] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., & Zagorodnov, D. (2009, May). *The eucalyptus open-source cloud-computing system*. In 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID'09. pp. 124-131. IEEE.

[11] He, Q., Zhou, S., Kobler, B., Duffy, D., & McGlynn, T. (2010, June). *Case study for running HPC applications in public clouds*. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. pp. 395-401. ACM.

[12] Bientinesi, P., Iakymchuk, R., & Napper, J. (2010). *HPC on competitive cloud resources*. In Handbook of Cloud Computing. pp. 493-516. Springer US.

[13] Brodkin, Jon. "$1,279-per-hour, 30,000-core Cluster Built on Amazon EC2 Cloud.", Ars Technica, 20 Sept. 2011. Website

[14] Bernstein, J. McMahon, K. (2012). *Computing on DemandHPC as a Service*. Penguin Computing

[15] Kureshi, Ibad. *The Queensgate Grid - Systems*, HPC Resource Centre RSS. HPC Research Group, University of Huddersfield, 04 Apr. 2010. http://hpc.hud.ac.uk/?page_id=446.

[16] Kureshi, Ibad. *The Queensgate Grid - Applications*, HPC Resource Centre RSS. HPC Research Group, University of Huddersfield, 04 Apr. 2010. http://hpc.hud.ac.uk/?page_id=175.

[17] Vujošević-Janičić, Milena, Nikolić, Mladen, Tošić, Dušan and Kuncak, Viktor (2013). *Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments.* Information and Software Technology, Volume 55, Number 6, pp.1004-1016.

[18] Distefano, Dino and Parkinson, Matthew J. (2008). *jStar: Towards Practical Verification for Java.* Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, OOPSLA '08, pp.213-226.

[19] Filliâtre, Jean-Christophe and Marché, Claude (2007). *The Why/Krakatoa/Caduceus Platform for Deductive Program Verification.* 19th International Conference on Computer Aided Verification. Lecture Notes in Computer Science, Volume 4590, pp.173-177. Springer.

[20] Evans, Eric (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison Wesley.

[21] Kureshi, I., Holmes, V., & Cooke, D. J. (2012, September). *Robust Mouldable Scheduling Using Application Benchmarking For Elastic Environments*. In Local Proceedings of the Fifth Balkan Conference in Informatics (No. 920, pp. 51-57). University of Novi Sad, Serbia.