Automatic Cloud Bursting under FermiCloud

Hao Wu*§, Shangping Ren*, Gabriele Garzoglio†, Steven Timm†, Gerard Bernabeu†,
Hyun Woo Kim†, Keith Chadwick†,Haengjin Jang‡, Seo-Young Noh‡

***Abstract*—Cloud computing is changing the infrastructure upon which scientific computing depends from supercomputers and distributed computing clusters to a more elastic cloud-based structure. The service-oriented focus and elasticity of clouds can not only facilitate technology needs of emerging business but also shorten response time and reduce operational costs of traditional scientific applications. Fermi National Accelerator Laboratory (Fermilab) is currently in the process of building its own private cloud, FermiCloud, which allows the existing grid infrastructure to use dynamically provisioned resources on FermiCloud to accommodate increased but dynamic computation demand from scientists in the domains of High Energy Physics (HEP) and other research areas. Cloud infrastructure also allows to increase a private cloud's resource capacity through "bursting" by borrowing or renting resources from other community or commercial clouds when needed. This paper introduces a joint project on building a cloud federation to support HEP applications between Fermi National Accelerator Laboratory and Korea Institution of Science and Technology Information, with technical contributions from the Illinois Institute of Technology. In particular, this paper presents two recent accomplishments of the joint project: (a) cloud bursting automation and (b) load balancer. Automatic cloud bursting allows computer resources to be dynamically reconfigured to meet users' demands. The load balance algorithm which the cloud bursting depends on decides when and where new resources need to be allocated. Our preliminary prototyping and experiments have shown promising success, yet, they also have opened new challenges to be studied.**

I. Introduction

Cloud computing is changing the infrastructure upon which scientific computing depends from supercomputers and distributed computing clusters to a more elastic cloud-based structure. Many research institutions and industrial companies are looking into merging or migrating their computer infrastructures to computer cloud where "unlimited" resources are available if needed. For instance, high energy physics (HEP) is one of the research areas that needs large computation infrastructure support. Experiments of HEP need large amount of computational resources for modeling and data analysis. Furthermore, data preservation in HEP becomes more and more critical since all those experiments need big data for the long term analysis [16], [17].

*Illinois Institute of Technology,10 W 31st street, 013, Chicago, IL, USA, {hwu28, ren}@iit.edu. The research is supported in part by NSF under grant number CAREER 0746643 and CNS 1018731.

†Fermi National Accelerator Laboratory, Batavia, IL, USA, {garzogli,timm,gerard1,hyunwoo,chadwick}@fnal.gov

‡National Institute of Supercomputing and Networking, Korea Institute of Science and Technology Information, Daejeon, Korea, {hjjang,rsyoung}@kisti.re.kr

§Hao Wu works as an intern in Fermi National Accelerator Laboratory, Batavia, IL, USA

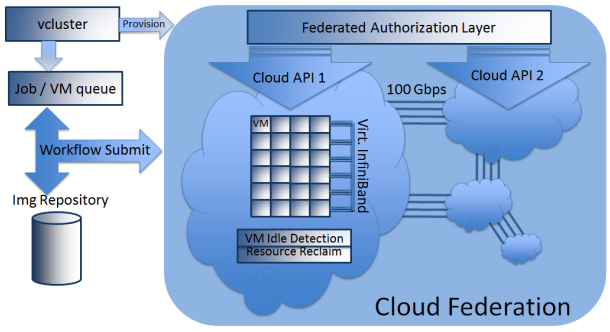

Fig. 1: High-level Architecture

Fermi National Accelerator Laboratory (Fermilab), as a leading research institution in the HEP filed, built a private computer cloud, the FermiCloud, in 2010. The FermiCloud is not only targeted for providing "infrastructure as a service (IaaS)" but also to provide higher-level cloud-hosted data movement and data processing services for scientific stakeholders as part of the longer-term strategic goal of providing "data processing as a service (DPaaS)" and other services to support the experiments in HEP. Since the establishment of the FermiCloud, the Fermilab Grid and Cloud Services department has smoothly integrated its grid computing infrastructure with the FermiCloud.

However, as the demands for computational resources from scientific applications keep increasing, the existing private FermiCloud infrastructure does not always meet the needs. Hence, Fermilab and Korea Institute of Science Technology and Information (KISTI) global science experimental data hub center started a joint project to build a cloud federation among the FermiCloud, KISTI's private cloud "GCloud", and commercial clouds, such as EC2 [1] and Microsoft Windows Azure [4], for scientific applications. Fig. 1 illustrates the high level architecture for the proposed cloud federation. The proposed cloud federation aims to enable scientific workflows of stakeholders operating on multiple cloud resources through (a) virtual infrastructure automation and provisioning, (b) interoperability and federation of cloud resources, and (c) high-throughput fabric virtualization.

The project has made significant progress through the collaboration between Fermilab and KISTI. In particular, we have successfully implemented cloud bursting which allows grid worker virtual machines to run on the cloud in response to demand for grid jobs. We have also implemented X.509 authentication [6] tool. This paper presents two recent accomplishments for the cloud bursting: (a) cloud bursting automation and (b) a load balancer. With the implementation of cloud bursting automation, virtual machines can be automatically launched according to batch job running status.

The virtual work machines can not only be launched on private cloud (FermiCloud, and GCloud), they can also be launched on public commercial clouds such as Amazon EC2 and Windows Azure when all the resources in the private clouds are consumed. The load balancer is the core component of the cloud bursting automation, it decides when and where a virtual work machine should be launched to.

The rest of the paper is organized as follows: Section II discusses related work. In Section III, we introduce the automatic cloud bursting architecture and its implementation details. Section IV presents a load balancing algorithm. Section V discusses the experimental results and points out our findings. We conclude and point out future work in Section VI.

II. Related Work

Cloud services are currently among the top-ranked high growth areas in computer services and seeing an acceleration in enterprise adoption with the worldwide market predicted to reach more than $140b in 2014 [14], [3]. The "pay-as-you-go" business model and the service oriented models allow the user to have "unlimited" resources if needed and free from infrastructure maintenance and software upgrades. Researchers from different areas have already started integrating and migrating their applications from computer grids to computer clouds, such research areas include life science [12], astronomy [15] and earthquake research [10], to name a few.

However, private clouds often have limited resources and may not always be able to meet the demand of increasingly large scientific applications. Juve *et al.* have shown the performance and operation costs of executing scientific applications on commercial clouds. The data have indicated that both performance and the costs are all in an acceptable range [10], [15]. In other words, using commercial clouds as external resources and build hybrid cloud to support the execution of resource demanding applications is a viable solution.

One successful example is the STAR project at the Brookhaven National Laboratory's Relativistic Heavy-Ion Collider [2] [5]. The STAR experiment studies fundamental properties of nuclear matter as it exists in a high-density state called a Quark Gluon Plasma. Because of the resource shortage from the local grid service, the STAR team started to collaborate with the Nimbus team at Argonne National Laboratory to migrate its experiment to a computer cloud. The Nimbus tools enable virtual machines in private cloud to be deployed on Amazon EC2.

Many researchers have made significant contributions on building the cloud federation. Celesti *et al.* proposed a three phase (discovery, match-making, and authentication) model and described an architectural solution called Cross-Cloud Federation Manager (CCFM) for cloud federation in [8]. Sotomayor *et al.* introduced their work for virtual infrastructure management in private and hybrid clouds [13]. In particular, they use OpenNebula as their cloud platform and combine with Haizea, a resource lease manager that act as a back-end scheduler for OpenNebula, to deploy virtualized services on both a local pool of resources and on external clouds. Cloud Scheduler [7], [9], a valuable cloud federation solution for the virtual batch job system designed for high energy physics, was presented by Armstrong et al. in 2010. They use HTCondor to build a virtual batch system. The whole virtual batch system is built above four different IaaS clouds, i.e. Nimbus, OpenNebula, Eucalyptus and Amazon EC2. The cloud scheduler launches predefined virtual machines on the four different clouds based on current status of Condor jobs.

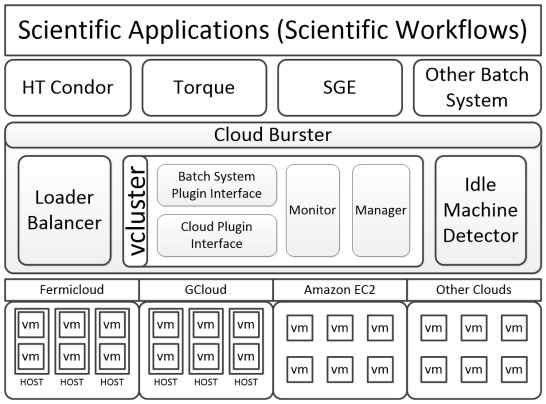

Fig. 2: Cloud Burster Architecture

Different from existing work in the literature, the major goals of automatic cloud bursting under the FermiCloud are to (1) provide an elastic cloud service for a variety of dynamic scientific applications, including computation intensive and interaction intensive applications, (2) make the cloud locations where applications are deployed to transparent to application users, and (3) improve application performance while reducing total operational cost. In the next section, we discuss the architecture of the cloud burster, a middleware that bridges the IaaS cloud providers and users, for automatic cloud bursting.

III. Cloud Bursting Architecture

In this section, we discuss the detailed design of the cloud burster middleware and the implementation of automatic cloud bursting. One of the goals of the joint project is to provide users an elastic virtual cluster to execute scientific applications with good performance. In order to achieve this goal, we have developed the cloud burster, a middleware that bridges multi-cloud IaaS providers and users. The cloud burster shields the cloud locations sources where different resources are provided from and provides upper level users one seemingly large resource pool. The cloud burster contains three main components: *vcluster*, idle machine detector and load balancer. Fig. 2 shows the the cloud burster architecture.

A. The vcluster

The functionality of the *vcluster* is to negotiate among different cloud providers. The *vcluster* also supports different types of batch systems, i.e. HTCondor, Torque, Sun Grid Engine (SGE) and other batch systems. For each of them, *vcluster* provides an "unlimited" resource pool through the underlying cloud federation. For the users, they can choose any of the batch systems with which they are familiar with and deploy their scientific applications on the batch job system.

The design of *vcluster* consists of four different components: batch system plugin interface, cloud plugin interface, monitor and manager.

Batch System Plugin Interface: The batch job system plugin interface is the module responsible for the communication between *vcluster* and batch job system. It retrieves work load information from batch job system, hence the cloud burster can do the resource provisioning accordingly. Since the interface is plugin based, different types of batch job systems can easily joint the *vcluster* at run time. However, each batch system is independent and does not communicate with each other.

Cloud Plugin Interface: The cloud plugin interface has similar design as the batch system plugin interface. It allows different IaaS clouds to join *vcluster* dynamically. The interface is responsible for the communication between the *vcluster* and different clouds. Unlike the independent batch system, all the cloud connected to *vcluster* are inter-connected. Currently, the FermiCloud and GCloud are both using OpenNebula as the IaaS platform. The cloud plugin interface has strong support to the OpenNebula. For the commercial cloud, the *vcluster* currently uses Amazon EC2 as its main external resource pool and uses REST API to communicate with Amazon EC2. However, it provides interfaces to other IaaS clouds, i.e. OpenStack, Nimbus etc. Many institutes and research facilities have expressed interest to collaborate in *vcluster*.

Monitor: The monitor module is used to collect information from both batch systems and clouds. Through the batch system plugin interface, it collects the job execution and waiting status on the batch systems. From the cloud plugin interface, it collects virtual machine operation status on different clouds. It also collects the physical machine status on different private clouds, i.e. utilization, virtual machine deployment etc.

Manager: The manager module is used to manage the plugins, batch systems, virtual machines and clouds. For new plugins of both batch system and cloud, the manager module registers them in the *vcluster* so that other modules can use the plugins and communicate with new registered batch system or cloud. The manager module controls each batch system and their under-layer resources from different cloud. The manager ensures that no resources are shared by different batch systems. Addition to the batch system and cloud plugin, *vcluster* also provides other plugins supports, such as load balancer and idle machine detector. And all those additional plugins are managed by the manager module. The manager module launches virtual machines, stops virtual machines and suspends virtual machines on different clouds according to the load balance decision made by load balancer module. A more detailed conceptual design of *vcluster* can be found in [11].

B. Idle Machine Detector

The idle machine detector component is used to detect idle virtual machines instantiated on a cloud and associated with one of the batch systems. The idle machine detection is a critical task in the cloud bursting. The system cannot always burst its resources, when there are idle virtual machines in the resource pool, we want to release those resources. However, accurate detection of idle virtual machine is challenging. When all the virtual machines are only used as work machines in one of the batch system, idle virtual machine detection can by achieved by retrieving the batch system status. However, such information given by batch system is not always reliable. For instance, as one batch system only monitors its own daemon that running on the virtual machine. The idle status does not mean the virtual machine is actually idle. The accurate idle virtual machine detection should consider CPU, disk IO, network IO etc. and need further study.

C. Load Balancer

Load balancer is the core component in the cloud burster, it determines when to start or terminate a virtual machine for the batch systems. With the load balancer, cloud burster frees users from resource provisioning and batch system maintenance, as it automatically allocates new resources for the batch systems if needed by scientific applications. On the other hand, if just few scientific applications are running on the batch job system, the load balancer has the capability to make decision of releasing the idle resources. The load balancer receives information from the monitor module. Combining both batch job status and cloud pool status, the load balancer performs a load balancing algorithm to decide if more resources are needed to accomplish the jobs in a batch system.

D. Cloud Bursting Automation

With the support of the *vcluster*, the load balancer and the idle machine detector, we are able to automatize the process of cloud bursting. Fig. 3 illustrates the workflow of cloud bursting automation process. The cloud burster first retrieves batch job status from the batch system plugin interface. If there is no job in the job queue, the cloud burster will determine whether there exits idle virtual machine. If there are idle virtual machine exist, the cloud burster will terminate those virtual machines and release the corresponding resources. The cloud burster repeats the procedure periodically with a fixed period.

If the batch system job queue is not empty, the cloud burster decides whether a new virtual machine is needed to execute waiting jobs. The cloud burster also collects the current virtual machine status from both private and commercial cloud and physical machines from the private cloud. Combining with the job status and virtual machine, physical machine status, the load balancer will decide whether new virtual machines are needed. If new virtual machines are needed, the manager module will launch new virtual machines on the corresponding IaaS clouds. The cloud burster will then wait and repeat the whole procedure until the new virtual machines are launched and ready to execute jobs. The detailed load balancing algorithm will introduced in the following section.

IV. Load Balancer

In this section, we first discuss some of the challenges of designing the load balancing algorithm. Then we introduce a load balancing algorithm for the cloud bursting automation.

A. Design Challenges

As mentioned in previous sections, load balancing is the most important component in the cloud bursting automation process. The design of the load balancing algorithm will directly impact the performance, energy consumption and operational cost of the whole system. There are three main challenges of designing an efficient load balancing algorithm.

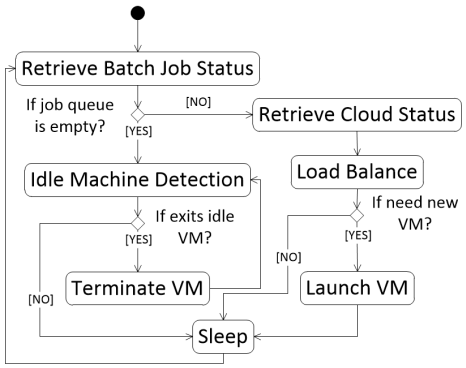

Fig. 3: Cloud Bursting Automation Process

Virtual machine launching overhead: Virtual machine launching overhead is the time it takes to launch a virtual machine and prepare the virtual machine for executing batch jobs. Such overhead can cause serious problems if it is not considered by the load balancing algorithm. For instance, if a new virtual machine is launched for executing jobs in the job queue, but some of the running jobs are completed during the time when a new virtual machine is launching. When the new virtual machine is ready to execute jobs, there is no jobs in the queue to be executed. Hence, the new virtual machine become a idle resource, which not only consumes power and other operational costs, but also increases the workload for both monitor module and idle machine detector model to reclaim back the resources allocated to the idle virtual machine.

Predicted batch job complete time : Predicted batch job complete time is the time predicted by a mechanism for when a batch job will complete its execution. Whether a new virtual machine needs to be launched depends on job predicted completion time. However, due to the dynamic nature of the scientific jobs and under layer resources, it is difficult to give an accurate prediction about the completion time of current jobs. Such inaccuracy can cause resource waste and increased operational costs if the completion time is overestimated.

Resource allocation strategy: A strategy that decides the physical devices where a virtual machine is allocated. In private clouds, such as OpenNebula, the default resource allocation strategy is to allocate new virtual machine on the least utilized physical machine. We denote this strategy as horizontal deployment. The horizontal deployment takes the virtual machine's performance as the priority. However, this strategy may lead to a low system utilization and consume more energy. On the contrary to horizontal deployment, there is another way to deploy virtual machines - vertical deployment. This method tries to deploy virtual machines on the same physical machine. By using the least physical machines, the system energy consumption can be reduced. However, with the vertical deployment strategy, the virtual machine performance may reduced since all virtual machines are squeezed together. Furthermore, when the virtual machines are being deployed on to commercial cloud, the cost also need to be considered. Hence, how to balance the system energy consumption, virtual machine performance and operational cost become a critical problem when design load balancing algorithm.

B. Load Balancing Algorithm

Taking all three challenges into consideration, we propose the load balancing algorithm. The first critical factor that considered in our load balancing algorithm design is the resource allocation strategy. Instead of the horizontal allocation method that OpenNebula provides by default, we use vertical allocation strategy. In other words, we always dispatch virtual machines on the physical machine that has the highest utilization but still has enough resources for the requested virtual machine. Doing so can reduced the total physical machines used for the cloud burster and hence reduce system total energy consumption and operational cost. On the other hand, we consider commercial cloud, i.e. Amazon EC2 as a large physical cluster with infinite capacity. Hence, Amazon EC2 always has the lowest priority when load balancer chooses physical machines. Only when all the resources in private cloud are consumed, virtual machines are deployed on Amazon EC2.

It is possible that some jobs are completed when the virtual machine is booting. Hence we consider the virtual machine launching overhead as another major factor in the algorithm. Currently, we only use a simple job complete time prediction strategy which is taking average job execution time as the reference to predict the job complete time. By comparing job average execution time and virtual machine launching overhead, the load balancer decide whether extra virtual machines are needed. In particular, if a running job's current execution time is smaller than the average job execution time and the gap is larger than the virtual machine launching overhead, we consider such a running job will not finish even if a new virtual machine is launched and ready to execute batch jobs. If a running job's current execution time is smaller than the average job execution time but the gap is smaller than the virtual machine launch overhead, or the job's current execution time is larger than the average job execution time, we consider such job will finish before a new virtual machine is ready to use. Then we count the number of running jobs that will finish soon. Comparing the number of waiting jobs, we estimate how many more virtual machines we needed to execute the jobs.

Algorithm 1 gives the detailed design of load balancing algorithm. Line 1 indicates that only when there are jobs in the job queue, the load balancing algorithm will decide whether extra virtual machine needed. Line 2 to line 10 implements that when there is no virtual machine running, load balancing will direct launch new virtual machine for the batch system. Line 12 to line 19 determine how many extra virtual machines are needed. Line 20 to line 27 launch virtual machines according to the number obtained from previous steps.

V. Experiments

We set up several experiments to test the virtual machine launching overhead and the performance of the load balancer.

A. Virtual Machine Launch Overhead

We first test the virtual machine launch overhead in FermiCloud. All the virtual machines that we launch on FermiCloud are configured as one CPU and 2GB memory with image size 15GB. We launch virtual machines on the FermiCloud hosts by the default OpenNebula policies. Fig. 4 depicts the launching overhead for each virtual machine we launched. We also test

Algorithm 1: Load Balance Algorithm

Input : Running job set R_j, Waiting job set W_j, Running VM set R_{VM}, Host set H and average job running time T_{ave}

```
if |W_j| ≠ 0 then
    if |R_VM| = 0 then
        host ← first host in H;
        for i ← 0 to |H| do
            if h_i has largest utilization and h_i can still
            allocate resource for new VM then
                host ← h_i
            end
        end
        deploy a new vm on host;
    end
    else
        i ← 0;
        forall the job j in R_j do
            T_j ← j's running time;
            if (T_ave − T_j) is larger than VM' launch
            overhead then
                i++
            end
        end
        for j ← 0 to i do
            host ← first host in H;
            for k ← 0 to |H| do
                if h_k has largest utilization and h_k can still
                allocate resource for new VM then
                    host ← h_k
                end
            end
            deploy a new vm on host;
        end
    end
end
```

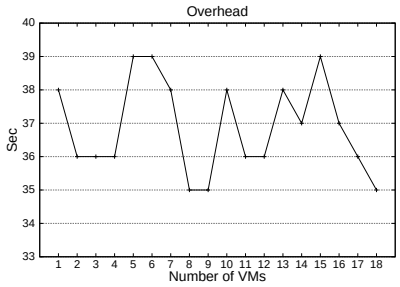

Fig. 4: VM Launching Overhead on FermiCloud

the virtual machine launch overhead on Amazon EC2. We use the micro instance as the test virtual machine. We launch virtual machines on Amazon's northern Virginia datacenter in different time period on daily base. The result is depicted in Fig. 5. It is not hard to tell from Fig. 5 that on Amazon EC2, virtual machine launch overheads are almost around

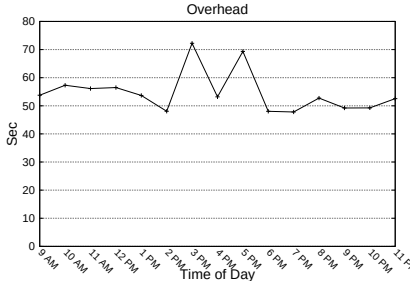

Fig. 5: VM Launching Overhead on Amazon EC2

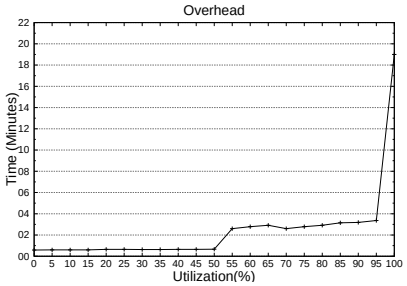

Fig. 6: Utilization Impact of VM Launch Overhead

one minute (55 seconds on average). However, variation of the launching overheads on Amazon EC2 is significant (44% compared with the average launching overhead). On the other hand, launching a virtual machine on FermiCloud takes 37 seconds on average, which is 18 seconds faster than launching a virtual machine on Amazon EC2. Furthermore, the launch overhead are quite stable on FermiCloud, all virtual machines are ready within the time range of 30 seconds to 40 seconds. The launching overhead variation on FermiCloud is only 11%. In addition, the virtual machine instance we launched on Amazon EC2 is micro instance, it may take longer time to launch more powerful virtual machines on Amazon EC2. This results may impact the performance of the cloud bursting.

As we discussed in section IV, however, performance of the virtual machine may suffer some degree of drop as the physical machine's utilization increase. We set up another experiment to see how the physical machine's utilization affect the virtual machine launching overhead. We first launch one virtual machine on one empty physical machine (no virtual machine has been deployed on it). After the virtual machine is successfully running on that physical machine, we let the virtual machine execute some tasks, i.e. disk write and read. Then we launch another virtual machine on the same physical machine. We repeat the process until no virtual machine can be deployed on the same physical machine. Fig. 6 shows the virtual machine launching overhead on the same physical machine as the physical machine's utilization increases.

As shown in Fig. 6, the virtual machine launching overheads are relatively small and steady when the physical machine is lightly utilized (utilization below 50 %). When the physical machine's utilization increases to above 55 %, the virtual machine launching overhead increases significantly, but remain the same level until the physical machine is almost fully utilized. The physical machine can still allocate resources for the new virtual machine when it has a relatively high utilization(more than 90%) , the overhead of launching a virtual machine becomes extremely large.

B. Performance of Load Balancer

In order to test the implementation of our cloud bursting automation and the performance of the load balancer, we conducted another set of experiments. We use HTCondor as our batch job system, and establish a virtual condor system using cloud burster. We create a set of simple jobs that only consume times but do nothing. We first run all the jobs using one single virtual machine. Then we run the same set of jobs using two virtual machines. At last, we run the same set of jobs without any virtual work machine and let cloud bursting

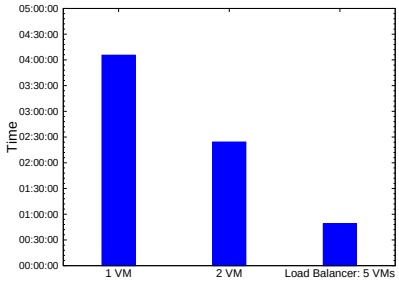

Fig. 7: Job Execution Time with Load Balancer

automation to decide the number of virtual machine needed to execute the jobs.

Fig. 7 illustrates the results. With only one virtual work machine, the total execution time for all the jobs is about four hours. When we adopt two virtual work machines, the total execution time reduces to almost half of the execution time using only one machine. However, with cloud bursting automation, the total execution time is reduced to 48 minutes with 5 virtual machines. After all the jobs are completed, the cloud burster automatically removes these virtual machines. This indicate the cloud bursting automation can successfully launch virtual work machines according to the batch job status. Furthermore, the results shows that, the load balancer can actually help the system to utilize the resources and hence to reduce the total execution time for the batch job system.

VI. Conclusion

This paper has presented an international collaboration project between Fermi National Accelerator Laboratory (Fermilab) and Korea Institute of Science and Technology Information (KISTI), with technical contributions from the Illinois Institute of Technology (IIT). The joint project is aimed to build a cloud federation for scientific applications, especially for the scientific workflows in the field of high energy physics. In particular, the proposed cloud federation aims to enable scientific workflows of stakeholders to run on multiple cloud resources by use of (a) virtual Infrastructure Automation and Provisioning, (b) interoperability and federation of cloud Resources, and (c) high-throughput fabric virtualization. This paper mainly focused on accomplishing one of the three objectives of cloud bursting automation and resource provisioning on the cloud federation. In the paper, we describe the design of cloud burster, a middleware that bridges the underlayer clouds and upper layer batch systems. We also show the implementation of cloud bursting automation and present an efficient load balancing algorithm to support the automation. Experiments results show that our cloud bursting automation is successfully running on the cloud federation and the load balancer can efficiently reduce the batch job total execution time. Furthermore, the FermiCloud is more efficient and stable on launching virtual machines compared with Amazon EC2.

Our preliminary prototyping and experiments have shown promising success, yet, they also have opened new challenges to be studied. From the experiment we know that the virtual machine launching overhead may vary in private clouds, when the physical machine has a high utilization, the overhead increases significantly. In addition, launching virtual machines on commercial cloud takes much more time than on FermiCloud. Hence further tests on virtual machines' performance on commercial clouds needed to be done in the future. Furthermore, the load balancing algorithm we proposed in the paper is simple. The algorithm can be improved by adopting more accurate job completion prediction heuristic algorithm. These will affect the load balancing algorithm on virtual machine deployment. In parallel, we will keep working on the project to achieve high-throughput fabric virtualization and interoperability and federation of cloud resources.

VII. Acknowledgement

This work is supported by the US Department of Energy under contract number DE-AC02-07CH11359 and by KISTI under a joint Cooperative Research and Development Agreement CRADA-FRA 2013-0001 / KISTI-C13013.

References

[1] Amazon ec2. aws.amazon.com.

[2] Feature - clouds make way for STAR to shine. http://www.isgtw.org/feature/isgtw-feature-clouds-make-way-star-shine.

[3] Icpads workshop on cloud services and systems. http://datam.i2r.a-star.edu.sg/css13/.

[4] Microsoft windows azure. http://www.windowsazure.com/en-us/.

[5] Nimbus and cloud computing meet STAR production demands. http://www.hpcwire.com/hpcwire/2009-04-02/nimbus_and_cloud_computing_meet_star_production_demands.html.

[6] Fermicloud, 2013. http://fclweb.fnal.gov.

[7] P. Armstrong, A. Agarwal, A. Bishop, A. Charbonneau, R. Desmarais, K. Fransham, N. Hill, I. Gable, S. Gaudet, S. Goliath, et al. Cloud scheduler: a resource manager for distributed compute clouds. *arXiv preprint arXiv:1007.0050*, 2010.

[8] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How to enhance cloud architectures to enable cross-federation. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 337–345. IEEE, 2010.

[9] I. Gable, A. Agarwal, M. Anderson, P. Armstrong, K. Fransham, D. H. C. Leavett-Brown, M. Paterson, D. Penfold-Brown, R. Sobie, M. Vliet, et al. A batch system for hep applications on a distributed iaas cloud. In *Journal of Physics: Conference Series*, volume 331, page 062010. IOP Publishing, 2011.

[10] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Scientific workflow applications on amazon ec2. In *E-Science Workshops, 2009 5th IEEE International Conference on*, pages 59–66. IEEE, 2009.

[11] S.-Y. Noh, S. C. Timm, and H. Jang. vcluster: A framework for auto scalable virtual cluster system in heterogeneous clouds. *Cluster Computing*. To appear.

[12] J. Qiu, J. Ekanayake, T. Gunarathne, J. Y. Choi, S.-H. Bae, H. Li, B. Zhang, T.-L. Wu, Y. Ruan, S. Ekanayake, et al. Hybrid cloud and cluster computing paradigms for life science applications. *BMC bioinformatics*, 11(Suppl 12):S3, 2010.

[13] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *Internet Computing, IEEE*, 13(5):14–22, 2009.

[14] TheCloudMarket.com. The Cloud Market Statistic Monitor, 2012. http://thecloudmarket.com.

[15] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, pages 15–24. ACM, 2011.

[16] D. A. e. a. Z. Akopov, Silvia Amerio. Data preservation in high energy physics, 2009. arXiv:0912.0255.

[17] D. A. e. a. Z. Akopov, Silvia Amerio. Status report of the dphep study group: Towards a global effort for sustainable data preservation in high energy physics, 2012. arXiv:1205.4667.