Overhead-Aware-Best-Fit (OABF) Resource Allocation Algorithm for Minimizing VM Launching Overhead

Hao Wu[*]
Illinois Institute of Technology
10 w 31 St.
Chicago, IL, 60616
hwu28@hawk.iit.edu

Shangping Ren[†]
Illinois Institute of Technology
10 w 31 St.
Chicago, IL, 60616
ren@iit.edu

Steven Timm[‡]
Fermi National Accelerator Laboratory
Batavia, IL, USA
timm@fnal.gov

Gabriele Garzoglio
Fermi National Accelerator Laboratory
Batavia, IL, USA
garzogli@fnal.gov

Seo-Young Noh[§]
National Institute of Supercomputing and Networking,
Korea Institute of Science and Technology Information
Daejeon, Korea
rsyoung@kisti.re.kr

ABSTRACT

FermiCloud is a private cloud developed in Fermi National Accelerator Laboratory to provide elastic and on-demand resources for different scientific research experiments. The design goal of the FermiCloud is to automatically allocate resources for different scientific applications so that the QoS required by these applications is met and the operational cost of the FermiCloud is minimized. Our earlier research shows that VM launching overhead has large variations. If such variations are not taken into consideration when making resource allocation decisions, it may lead to poor performance and resource waste. In this paper, we show how we may use an VM launching overhead reference model to minimize VM launching overhead. In particular, we first present a training algorithm that automatically tunes a given reference model to accurately reflect FermiCloud environment. Based on the tuned reference model for virtual machine launching overhead, we develop an overhead-aware-best-fit resource allocation algorithm that decides *where* and *when* to allocate resources so that the average virtual machine launching overhead is minimized. The experimental results indicate that the developed overhead-aware-best-fit resource allocation algorithm can significantly improved the VM launching time when large number of VMs are simultaneously launched.

[*]Hao Wu works as an intern in Fermi National Accelerator Laboratory, Batavia, IL, USA

[†]The research is supported in part by NSF under grant number CAREER 0746643 and CNS 1018731.

[‡]This work is supported by the US Department of Energy under contract number DE-AC02-07CH11359

[§]This work is supported by KISTI under a joint Cooperative Research and Development Agreement CRADA-FRA 2013-0001 / KISTI-C13013.

1. INTRODUCTION

Because of its elasticity and flexibility, cloud technology has not only benefited general purpose computing we encounter in our daily life, such as Gmail, Google docs, iCloud, to name a few. It also brings new opportunities to scientific applications. For instance, the Nimbus team at Argonne National Laboratory successfully migrates the STAR experiment at the Brookhaven National Laboratory to Amazon EC2 to avoid the shortage from local grid service [2]. Another successful example of deploying scientific applications on computer clouds is the ATLAS experiment at the Large Hadron Collider at CERN which uses Google Cloud to improve the efficiency of its research process [4].

One of the main advantages of deploying scientific applications on computer cloud is that computer cloud has "infinite" amount of resources. Scientific applications often require large amount computational resources and have long execution time. With a traditional grid system, if local resources are fully occupied by some applications, the newly arrived applications have to wait until one of the running applications finishes and releases its resources. However, with the cloud technology, even if local resources are fully occupied, public cloud resources are always available to execute newly arrived applications.

Many institutions and companies have already foreseen the advantages of deploying scientific applications on cloud and have developed their specific cloud services for scientific applications. For instance, Amazon provides a HPC cloud for scientific applications [1], Microsoft [3] and Google [4] also provide specific cloud services for scientific applications.

Fermi National Laboratory (Fermilab), as a leading physics research institution in United States, has developed its own private cloud – the FermiCloud to support scientific applications within Fermilab and its collaboration institutions. Since the establishment of the FermiCloud in 2010, the Fermilab Grid and Cloud Services department has smoothly integrated its grid computing infrastructure with the FermiCloud. The FermiCloud project has made significant progress through the collaboration between Fermilab, Korea Institute of Science Technology and Information (KISTI) global science experimental data hub center and Illinois Institute of Technology(IIT). In particular, we have successfully implemented an automation tool the "*vcluster*", which allows grid worker virtual machines to run on the cloud in response to increased demand of grid jobs [11, 13]. The design goal of the *vcluster* is to automatically provisioning resources for different scientific applications so that the QoS of the scientific application is met and the operational cost of the FermiCloud is minimized.

Many solutions have been proposed by researchers to achieve the objectives that are similar to the FermiCloud design goal. The research includes minimizing application's makespan in cloud [10], reducing energy consumption of datacenters [12], minimizing cost of the application execution [8], to name a few. However, these researches assume that virtual machine launching overhead is a constant or even negligible. As Mao *et al.* pointed out that the virtual machine launching overhead can have a very large variation in public cloud [9]. Without considering the variations of virtual machine launching overhead when designing resource allocation algorithm in cloud may lead to cost increase and resource waste.

Our earlier work has studied virtual machine launching overhead under different system conditions and developed a reference model to predict virtual machine launching overhead [14]. However, as each physical host machine has its own characteristics. Even two host machines that have the same configuration may differ in performances. In order to obtain accurate predictions, the parameters of the reference model need to be adjusted for each host machine. In this paper, we first present a training algorithm that automatically tunes the accuracy of the virtual machine launching overhead reference model for a given physical platform. Based on the tuned virtual machine launching overhead reference model, we present an overhead-aware-best-fit (OABF) resource allocation algorithm that decides *where* and *when* to allocate resources so that the average virtual machine launching overhead is minimized is proposed.

The rest of the paper is organized as follows: In Section 2, we present our prior work regarding virtual machine launching overhead and the software we have developed for virtual resource management . Section 3 discuss the details of the automatic model training algorithm. The OABF algorithm is presented in Section 4. Section 5 discusses the experimental results. We conclude our work in Section 6.

2. PRELIMINARY WORK

2.1 The *vcluster* System

The *vcluster* is a middleware jointly developed by KISTI and Fermilab for automatically managing virtual resources according based on batch job load in system [11, 13]. It consists of four major components: batch system plugin interface, cloud plugin interface, monitoring module and load balancer plugin interface. The batch system plugin interface provides supports for communicating with different batch job systems such as HTCondor, Torque, and Sun Grid Engine (SGE). The cloud plugin interface is responsible for communicating with different cloud platforms such as FermiCloud in Fermilab, GCloud in KISTI, and public clouds such as Amazon EC2. The monitoring module collects all the information from connected batch job systems and cloud platforms. It translates the data from different batch job systems and cloud platforms into a uniform data format that can be used by load balancer interface. With the design of plugable modules for different batch job system, cloud platforms and load balancers, the *vcluster* can be easily extended and modified if needed.

The *vcluster* has been successfully implemented on FermiCloud GCloud and Amazon EC2 [11, 13]. The source code of the *vcluster* can be found in [6, 5]. In this paper, we present the design of resource allocation mechanism used in the *vcluster*'s load balancer module which decide *when* and *where* to deploy virtual machines so that the average virtual machine launching overhead is minimized.

2.2 VM Launching Overhead Reference Model

Our early study reveals that virtual machine launching overhead has large variations under different system conditions. Based on large set of experiments conducted under operational FermiCloud [14], we have established a virtual machine launching overhead reference model. It models the CPU utilization overhead and IO utilization overhead during the process of launching a virtual machine. The CPU utilization overhead consists of two main parts. One is at the virtual machine image transferring time:

$$U_{T_t}(t) = \frac{1}{1+e^{-0.5(T_t+t)(t-t_r)}} - \frac{1}{1+e^{-0.5(T_t+t_r)(t-(T_t+t_r))}} \tag{1}$$

where T_t is the image transferring time and t_r is the virtual machine release time. The image transferring time is impacted by the system IO utilization and network bandwidth.

The second part of CPU utilization overhead is at the virtual machine booting time:

$$U_b(t) = c * \frac{1}{m} e^{-\gamma(1-IO_s(h,t-1))(t-T_t))} \tag{2}$$

where c and γ are two system configuration dependent constants, m is the number of cores on the host machine and $IO_s(h,t)$ represents the system's disk IO utilization at time t. The CPU utilization overhead is also impacted by the IO utilization.

The main IO utilization overhead occurs at the time when virtual machine image is transferred. The overhead is proportional to the ratio of available IO bandwidth to the total IO bandwidth on the physical host machine. The virtual machine launching time t_b can be predicted based on the CPU utilization overhead, i.e.,

$$t_b = \max\{t || U_b'(t) |\leq \epsilon\} \tag{3}$$

where $U_b'(t)$ is the first derivative of $U_b(t)$ and ϵ is the threshold to determine whether the virtual machine's CPU utilization consumption become stable. The virtual machine launch time is calculated by adding image transfer time and boot time.

In [14], we have empirically shown that the developed reference model can accurate predict the virtual machine launching overhead. However, under different systems, the parameters of the model may change. In the next section, we present a training mechanism that can automatically adjust the parameters of the model to accurately reflect a given physical platform.

3. CALIBRATING VM LAUNCHING OVERHEAD REFERENCE MODEL ON FERMICLOUD PLATFORM

3.1 Prediction Accuracy Evaluation

There are many different benchmark methods to evaluate the accuracy of predictions. One of the most widely used approach is to compared the absolute error between the actual data and predicted data. Benchmarks such as mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE), and median absolute error (MdAE) are the different ways to measure the absolute errors. However, as pointed by Hyndman *et al.*, one of the disadvantages the absolute error based measurements face is that they are scale-dependent [7].

Another category of measurement methods is based on the percentage error. In contrast to the absolute error based measurements, the advantage of percentage error based measurements is that they are scale-independent. The percentage error based measurement methods include mean absolute percentage error (MAPE), median absolute percentage error (MdAPE), root mean square percentage error (RMSPE), and root median square percentage error (RMdSPE). However, the percentage error based measurements do not come flawless either. One of the main concerns with the percentage error based measurements is that the error can be infinite or undefined if the actual value is close to zero or being zero [7].

To overcome the disadvantages of the measurement methods mentioned above, Hyndman *et al.* proposed a new measurement method, i.e., the Mean Absolute Scaled Error [7]. The idea of their method is to scale the absolute error based on the in-sample MAE obtained from a benchmark prediction method. The scaled error is defined as:

$$q_t = \frac{e_t}{\frac{1}{n-1}\sum_{i=2}^{n}|Y_i - Y_{i-1}|} \tag{4}$$

where, e_t is the absolute error between the actual data Y_t and predicted data F_t at time point t; and Y_i represents the actual data from i^{th} prediction. The Mean Absolute Scaled Error is:

$$\text{MASE} = \text{mean}(|q_t|) \tag{5}$$

In our model training mechanism, we use MASE to calibrate the VM launching overhead reference model for FermiCloud platforms. It is worth pointing out that though the paper focuses on FermiCloud platforms, the methodology developed in this paper can be applied to any physical cloud platforms.

3.2 Model Training Algorithm

The basic idea of the proposed training algorithm is to adjust the ϵ in equation(3) so that the mean absolute scaled error of the predicted VM launching time is maintained within a reasonable error range. The detailed algorithm is illustrated in Algorithm 1.

Each time when a new actual virtual machine launching time is read from the system, the training algorithm is executed to calibrate the accuracy of the virtual machine launching overhead model. As Line 1 to Line 4 in Algorithm 1 indicate, if the new MASE value is within the designed error range, no new calibration needs to be performed. Otherwise, a new ϵ needs to be calculated. We use an example to explain how to calculate the new ϵ.

Assume the calibrated value from last round training is $\epsilon = 0.032$. Intuitively, since all historical predictions are accurate with ϵ, when a new actual value is observed, if needed, just small calibration needs to be performed on ϵ to ensure MASE within the error range. In this case, the precision of ϵ is at thousandth level (calculated by Line 5 and obtain 0.001), then the calibration is performed at thousandth precision level. To start calibration, we first treat $\epsilon = 0.032$ as $\epsilon' = 0.030$ (Line 6) and use ϵ' as a middle start point. Then, we search the new ϵ above and below the ϵ' in a small range ($\pm 0.001 \times 10$). In this case, the search range is $[0.021, 0.039]$.

Next, the algorithm calculates the MASE of the each corresponding ϵ' from 0.021 to 0.039 and selects the ϵ' that has the smallest MASE(Line 10 to Line 30). If the smallest MASE is within the error range, the search is terminated. Otherwise, the algorithm increases the precision (Line 31) and repeat the search procedure. For instance, suppose the $\epsilon' = 0.033$, and the precision is increased to 0.0001. Since $\epsilon' = 0.033$ gives smallest MASE in the range $[0.021, 0.039]$, the desired ϵ must exist in the range of $(0.0320, 0.0340)$. The algorithm continues the search in the range of $(0.0320, 0.0340)$ starting from 0.0330 (Line 9 to Line 31). The search terminates when a ϵ' is found such that the MASE value calculated by it satisfies the error range. It is possible that the search never find such an ϵ'. Hence the other terminate condition for the algorithm is the precision reaches a predefined precision threshold.

4. OVERHEAD-AWARE-BEST-FIT RESOURCE ALLOCATION DESIGN AND IMPLEMENTATION

The previous section has discussed how we automatically calibrate the accuracy of the virtual machine launching overhead model. In this section, we present an overhead-aware-best-fit resource allocation algorithm, i.e., the OABF algorithm, that aims to reduce average virtual machine launching time and show an implementation of the OABF on the FermiCloud.

4.1 The OABF Algorithm

Our preliminary study [14] reveals that when multiple virtual machines are launched simultaneously, the VM launching time increases as the number of simultaneous launches increases. We also have noticed that when a virtual machine that being deployed uses the same image as the last virtual machine that has been deployed on the same host machine, the virtual machine takes less time to launched. In other words, the VM launching time is significantly reduced if memory cache can be used.

Algorithm 1: Model Training Algorithm

```
Input  : Threshold $\epsilon$, Predicted Time Set $T_P$, Actual Time
         Set $T_A$
Output: Threshold $\epsilon$
1  $e \leftarrow$ calculateMASE($T_P, T_A$)
2  if $e \leq ErrorThreshold$ then
3      return $\epsilon$
4  end
5  $g \leftarrow$ calculateCurrentPrecision($\epsilon$)
6  $\epsilon' \leftarrow \max\{\lfloor \frac{\epsilon}{g \times 10} \rfloor, 1\} \times g \times 10$
7  Recalculate predicted time set $T_P$ using $\epsilon'$
8  $e \leftarrow$ calculateMASE($T_P, T_A$)
9  do
10     for $i \leftarrow 1$ to $9$ do
11         $\epsilon'' \leftarrow \epsilon' - i \times g$
12         if $\epsilon'' \leq 0$ then
13             break
14         end
15         Recalculate predicted time set $T_P$ using $\epsilon''$
16         $e' \leftarrow$ calculateMASE($T_P, T_A$)
17         if $e' \leq e$ then
18             $e \leftarrow e'$
19             $\epsilon \leftarrow \epsilon''$
20         end
21     end
22     for $i \leftarrow 1$ to $9$ do
23         $\epsilon'' \leftarrow \epsilon' + i \times g$
24         Recalculate predicted time set $T_P$ using $\epsilon''$
25         $e' \leftarrow$ calculateMASE($T_P, T_A$)
26         if $e' \leq e$ then
27             $e \leftarrow e'$
28             $\epsilon \leftarrow \epsilon''$
29         end
30     end
31     $g \leftarrow g/10$
32 while $e \leq ErrorThreshold \vee g \leq PrecisionThreshold$;
33 return $\epsilon$
```

Based on the our previous experimental work and experimental observation, we develop an overhead-aware-best-fit resource allocation algorithm. The fundamental drive behind the algorithm is to avoid simultaneous launches and to deploy virtual machines with the same image on the same host machine sequentially. Algorithm 2 gives the pseudo code of the OABF algorithm.

Each virtual machine v in the system is characterized by its release time t_r, host h that v is deployed on and waiting time t_w that denotes the offset from its release time. Once a virtual machine is released, its release time is recorded but without any host and waiting information. By running Algorithm 2, the virtual machine is assigned to the host that is predicted to have the shortest launching time. Its waiting time that indicates actual deploy time point offsets from its release time is given.

In particular, the algorithm compares the predicted launch time for v when it is deployed on each of the hosts (Line 3 to Line 22). For each host, the algorithm calculates the virtual machine launching overhead using equation 1 and 2 and its predicted launch time using equation 3. Then it compares the predicted launch time for v when v starts at different time points (Line 9 to Line 21). As mentioned before, the intuition of the OABF algorithm is trying to avoid simultaneous launch, hence we only check the time points that the virtual machines that have been deploy on the same host before v have predicted to finish image trans-

Algorithm 2: overhead-aware-best-fit Algorithm

```
Input  : Empty Virtual Machine $v = \{t_r, h, t_w\}$, Host set
         $H = \{h_1, \dots, h_n\}$, VM waiting queue
         $Q = \{v_1^q, \dots, v_m^q\}$
Output: Virtual Machine $v = \{t_r, h, t_w\}$ with host and
         waiting time information
1  $v.h \leftarrow$ null; $v.t_w \leftarrow 0$; $h' \leftarrow null$
2  $t'_w \leftarrow 0$; $t_b \leftarrow \infty$; $t'_r \leftarrow v.t_r$
3  for $i \leftarrow 1$ to $n$ do
4      $v.t_r \leftarrow t'_r$
5      $t_p \leftarrow$ calculatePredictLaunchTime($h_i, v$)
6      if $t_p \leq t_b$ then
7          $t_b \leftarrow t_p$; $h' \leftarrow h_i$
8      end
9      for $j \leftarrow 1$ to $m$ do
10         if $v_j^q$ is deployed on $h_i$ then
11             $t_t \leftarrow v_j^q$'s predicted image transfer time
12             if $t_t + v_j^q.t_r + v_j^q.t_w \geq v.t_r$ then
13                 $v.t_r \leftarrow t_t + v_j^q.t_r + v_j^q.t_w$
14                 $t_p \leftarrow$ calculatePredictLaunchTime($h_i, v$)
15                 if $t_p + t_t + v_j^q.t_r + v_j^q.t_w - v.t_r \leq t_b$ then
16                     $t_b \leftarrow t_p$; $h' \leftarrow h_i$
17                     $t'_w \leftarrow t_t + v_j^q.t_r + v_j^q.t_w - v.t_r$
18                 end
19             end
20         end
21     end
22 end
23 $v.t_r \leftarrow t'_r$; $v.h \leftarrow h'$; $v.t_w \leftarrow t'_w$
24 return $v$
```

ferring process(Line 10 to Line 20). Finally, a best fit host h with shortest predicted virtual machine launching time and waiting time t_w are assigned to v (Line 23).

4.2 Implementation

The implementation of the resource allocation automation process is embedded into the load balancer module that in the *vcluster*. Figure 1 depicts the architecture of *load balancer* in the *vcluster*.

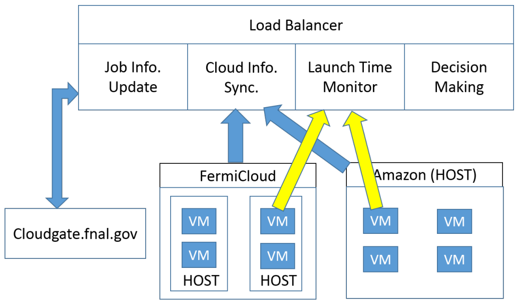

Figure 1: Architecture of Load Balancer

As shown in Figure 1, the load balancer has four submodules: job information update module, cloud information synchronization module, launch time monitoring module, and decision making module. The job information update module is responsible for fetching information from the batch job system. Since this paper focuses on reducing the virtual machine launching overhead, we skip the details of the job information update module.

The cloud information synchronization module is used to synchronize real time virtual machine information, host information and cloud platform information with the information predicted by load balancer. Due to the system error or manual operations, the information predicted and kept in load balancer may not be consistent with the real system information. Hence, the cloud information synchronization module is to check the consistence of stored predicted data and real system information, and ensure the correct information is provided to decision making process.

The launch time monitoring module is used to collect virtual machine's actual launching time. Once a virtual machine is actually launched and running, it reports the time stamp to the launch time monitoring module. After the launch time monitoring module receives the time stamp, it calculates the launch time for that virtual machine and record the time for the virtual machine launching overhead training process. In the system, there is a virtual machine waiting queue contains all the virtual machine that yet to be launched or the virtual machine under launching process. Once a virtual machine that in the waiting queue reports its actual launching time, the virtual machine is removed from the waiting queue.

The decision making module executes the resource allocation algorithm. It takes the information from other three modules to decide *where*, *when* and *what* to launch virtual machines. In this paper, we only focus on *when* and *where* to launch the virtual machine so that the average virtual machine launching overhead is minimized.

Figure 2 depicts the workflow of resource allocation automation process. It illustrates how different modules cooperate to automatically allocate resources and calibrate the virtual machine launching overhead reference model.

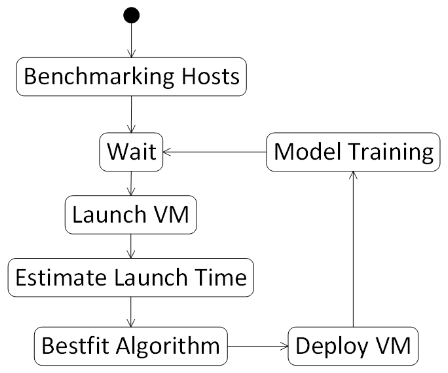

Figure 2: Resource Allocation Automation Workflow

When the load balancer starts, it first benchmarks all host machine's performance, i.e. disk I/O bandwidth, network bandwidth, etc. It then goes into waiting state. Once a virtual machine request is released, the load balancer needs to decide *when* and *where* to launch the virtual machine. Based on the virtual machine launching overhead reference model, a predicted launch time is calculated. The predicted launch time is adapted by OABF algorithm to determine the host machine where the virtual machine should be deployed on. Finally, the virtual machine is initialized and deployed on the assigned host machine. After the virtual machine is launched, the load balancer uses the actual launch time to calibrate the accuracy of the model using Algorithm 1.

5. EVALUATION

The overhead-aware-best-fit algorithm is evaluated under real cloud environment–FermiCloud. Since FermiCloud is designed for scientific applications and it is very likely that when an application that needs large amount resources is submitted to the system, large number of virtual machines are needed to be launched simultaneously. Hence, our evaluation focuses on the performance of the propose OABF algorithm under larger number of simultaneous launches.

5.1 Experiment Setting

The experiments are performed under FermiCloudẆe use total ten host machines for the experiment. All ten hosts are configured with 8-core Intel(R) Xeon(R) CPU X5355 @ 2.66GHz and 16GB memory. All these machines are connected through high speed Ethernet. We use OpenNebula as the cloud platform. The OpenNebula front end server has 16-core Intel(R) Xeon(R) CPU E5640 @ 2.67GHz, 48GB memory.

5.2 Launching Time Comparison

In order to evaluate the performance of the developed OABF algorithm, we compare the virtual machine launching time when the OpenNebula default scheduler is used with the virtual machine launching time when the OABF algorithm is used. Each time, we launch seventy (70) virtual machines simultaneously. All virtual machines are using the same image. Same set of contextualization scripts are used for launching VMs. The first experiment is to launch virtual machines using QEMU Copy On Write images (QCOW2) with size 2.6GB.

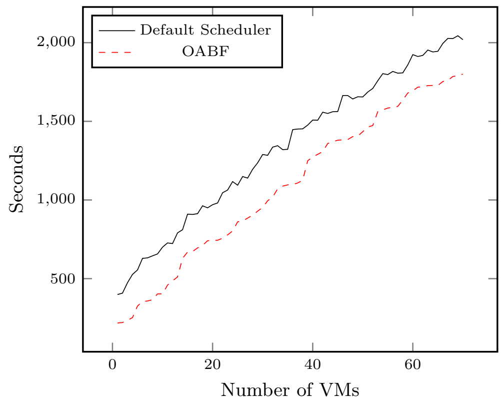

Figure 3: VM Launching Time Comparison using QCOW2 Image

Figure 3 depicts the comparison of virtual machine launching time with the OpenNebula default scheduler and virtual machine launching time with the OABF algorithm. All the virtual machines are ordered by their launching time in increasing order. From the Figure 3, it is clear that with the OABF algorithm, virtual machines take less time to launch than using OpenNebula default scheduler. The launching time reduction from the OABF is rather stable from the first virtual machine to the last virtual machine. The maximum reduction among the seventy virtual machine is 349 seconds. On average, the virtual machine launching time reduction is 245.44 seconds, which is more than four minutes reduction compared to the OpenNebula default scheduler.

The second experiment is to test the performance on large images. We also launch seventy (70) virtual machines at a

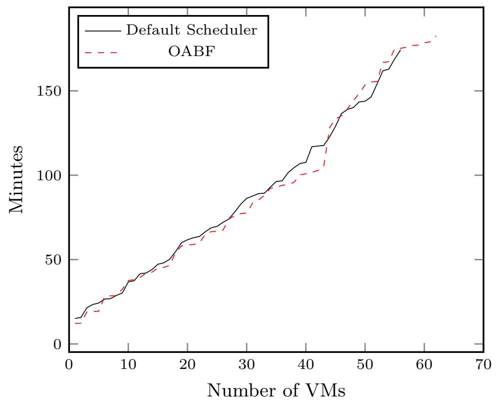

Figure 4: VM Launching Time Comparison using RAW image

time. Each virtual machine now use RAW image with size 16 GB. Figure 4 dispatches the comparison results between virtual machines' launching time with the OpenNebula default scheduler and virtual machines' launching time with the OABF algorithm. As shown in Figure 4, when the image size is large, the virtual machine launching time reduction from OABF can be clearly observed. The maximum reduction from OABF algorithm is 907 seconds which is about 15 minutes saving.

In our experiments, we have also observed that after 50 virtual machines being launched, the remaining virtual machines takes longer time to launch when using the OABF algorithm than with the OpenNebula default scheduler. This is because most of the remaining virtual machines (15 out of 20) are failed to launch under OpenNebula default scheduler due to large amount I/O operations, hence there are just few virtual machines are actually undergoing launching process. On the other hand, when using the OABF algorithm, only five virtual machines are failed to launch. The OABF algorithm not only reduces the virtual machines' launching times when large amount simultaneous launches occur but also can improve the success rate of such extreme scenario. With the OABF algorithm, on average, the virtual machine saves 103 seconds on launching process compared with the OpenNebula default scheduler.

6. CONCLUSION

The FermiCloud is a private cloud built in Fermi National Accelerator Laboratory to provide on-demand resources for different scientific applications. The design goal of FermiCloud is to automatically provision resources for different scientific applications so that the QoS of the scientific application is met and the operational cost of FermiCloud is minimized. The main challenge of designing the FermiCloud system is to decide *when* and *where* to allocate resources so that the goals are met. In this paper, we present a mechanism to automatically train the VM launching overhead reference model that we previously developed. Based on the virtual machine launching overhead reference model, we have developed in this paper an overhead-aware-best-fit resource allocation algorithm to help the cloud reduce the average VM launching time. In the paper, we have also presented an implementation of the developed OABF algorithm on FermiCloud. The experimental results indicate that the OABF can significantly reduce the VM launching time (reduced VM launch time by 4 minutes on average) when large number of VMs are launched simultaneously.

7. REFERENCES

[1] AWS HPC cloud computing. http://aws.amazon.com/hpc/.

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

[3] High performance computing on microsoft azure for scientific and technical applications. http://research.microsoft.com/en-us/projects/azure/high-perf-computing-on-windows-azure.pdf.

[4] Mapping the secrets of the universe with google compute engine. https://cloud.google.com/developers/articles/mapping-the-secrets-of-the-universe-with-google-compute-engine?hl=ja.

[5] Source code for FermiCloud version vcluster. https://github.com/philip-wu5/project/tree/fermi.

[6] Source code for KISTI version vcluster. https://github.com/vcluster/project.

[7] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[8] M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12. IEEE, 2011.

[9] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012.

[10] Y. Z. Mengxia Zhu, Qishi Wu. A cost-effective scheduling algorithm for scientific workflows in cloud. *Proceedings of 31st IEEE International Performance Computing and Communications Conference*, 2012.

[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] A. M. Sampaio and J. G. Barbosa. Optimizing energy-efficiency in high-available scientific cloud environments. In *Cloud and Green Computing (CGC), 2013 Third International Conference on*, pages 76–83. IEEE, 2013.

[13] H. Wu, S. Ren, G. Garzoglio, S. Timm, G. Bernabeu, H. W. Kimy, K. Chadwick, H. Jang, and S.-Y. Noh. Automatic cloud bursting under fermicloud. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 681–686. IEEE, 2013.

[14] H. Wu, S. Ren, G. Garzoglio, S. Timm, G. Bernabeu, and S.-Y. Noh. Modeling the virtual machine launching overhead under fermicloud. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 374–383. IEEE, 2014.