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
garzoglio@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.

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].

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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 20XX ACM X-XXXXXX-XX-XX/XX ...$15.00.

FERMILAB-CONF-14-467-CD
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 a 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 HTCCondor, 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 pluggable 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:

\[
U_{Tf}(t) = \frac{1}{1 + e^{a(\gamma(T_i + t) - t_f)}} - \frac{1}{1 + e^{a(\gamma(T_i + t_r) - (t_f + t_r))}}
\]

where \(T_i\) 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:

\[
U_b(t) = c \cdot \frac{1}{m^t} \cdot e^{-\gamma((1 - IO_s)(h, t - 1) - (t - T_i))}
\]

where \(c\) and \(\gamma\) 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 \(t_o\) can be predicted based on the CPU utilization overhead, i.e.,

\[
t_o = \max\{t | U_o(t) \leq \epsilon\}
\]

where \(U_o(t)\) is the first derivative of \(U_o(t)\) and \(\epsilon\) 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 accurately 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 compare 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 relative percentage error (MAPE), median relative percentage error (MdAPE), root mean square percentage error (RMSE), and root median square percentage error (RMedSPE). 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:

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

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

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

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 \(\epsilon\) 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 \(\epsilon\) needs to be calculated. We use an example to explain how to calculate the new \(\epsilon\).

Assume the calibrated value from last round training is \(\epsilon = 0.032\). Intuitively, since all historical predictions are accurate with \(\epsilon\), when a new actual value is observed, if needed, just small calibration needs to be performed on \(\epsilon\) to ensure MASE within the error range. In this case, the precision of \(\epsilon\) 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 \(\epsilon’\) as a middle start point. Then, we search the new \(\epsilon\) above and below the \(\epsilon’\) 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 \(\epsilon’\) from 0.021 to 0.039 and selects the \(\epsilon’\) 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 \(\epsilon\) 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 \(\epsilon’\) is found such that the MASE value calculated by it satisfies the error range. It is possible that the search never find such an \(\epsilon’\). 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.
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 transferring 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 sub-modules: 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 its predicted launch time using equation 3. Then it calculates the predicted MASE (Line 8), to indicate actual deploy time point offsets from its release time.
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 FermiCloud. We 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
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 applications 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