5-2013

Accelerating optimistic HLA-based simulations in virtual execution environments

Zengxiang LI

Xiaorong LI

Nguyen Binh Duong TA
Singapore Management University, donta@smu.edu.sg

Wentong CAI

Stephen John TURNER

Accelerating Optimistic HLA-based Simulations in Virtual Execution Environments

ZengXiang Li, Xiaorong Li, TA Nguyen Binh Duong
Institute of High Performance Computing
Agency for Science, Technology and Research (A*STAR) Singapore
{liz,lixr,tanbd}@ihpc.a-star.edu.sg

Wentong Cai, Stephen John Turner
School of Computer Engineering
Nanyang Technological University Singapore
{aswtcai,assjturner}@ntu.edu.sg

ABSTRACT

High Level Architecture (HLA)-based simulations employing optimistic synchronization allows federates to process event and to advance simulation time freely at the risk of over-optimistic execution and execution rollbacks. In this paper, an adaptive resource provisioning system is proposed to accelerate optimistic HLA-based simulations in *Virtual Execution Environment* (VEE). A performance monitor is introduced using a middleware approach to measure the performance of individual federates transparently to the simulation application. Based on the performance measurements, a resource manager distributes the available computational resources to the federates, making them advance simulation time with comparable speeds. Our proposed approach is evaluated using a real-world simulation model with various workload inputs and different parameter settings. The experimental results show that, compared with distributing resources evenly among federates, our proposed approach can accelerate the simulation execution significantly using the same amount of computational resources.

Categories and Subject Descriptors

I.6 [**Computing Methodologies**]: SIMULATION AND MODELING

Keywords

HLA-based Simulations, Optimistic Synchronization, Virtual Execution Environment, VM Scheduling, Cloud Computing

SIGSIM-PADS'13, May 19-22, 2013, Montréal, Québec, Canada.

1. INTRODUCTION

Parallel and distributed simulations are usually built to study large-scale and complex systems in desired detail and fidelity. The *High Level Architecture* (HLA), IEEE 1516 standard [11], provides a general framework to build a parallel and distributed simulation (*federation*) by integrating various simulation models (*federates*). Most work to date has focused on conducting HLA-based simulations on a traditional execution platform composed of a group of dedicated computing nodes. Recently, we have witnessed an increasing interest of moving data and computation to the Cloud. Large amount of resources can be obtained on the Cloud for the purpose of executing compute- and data-intensive HLA-based simulations [6]. Furthermore, resources can be provisioned elastically according to the simulation workload. Different from traditional execution platforms, most of the Infrastructure as a Service (IaaS) Cloud providers [1] adopt *Virtual Execution Environment* (VEE) relying on virtualization technologies [2]. In this paper, we investigate how to accelerate HLA-based simulations in VEE.

One of the greatest challenges in HLA-based simulations is the time synchronization among federates, which ensures that all events are processed in *time stamp* (TS) order. In this paper, we focus on the optimistic synchronization approach [13], which allows a federate to process events and to advance simulation time freely at the risk of over-optimistic executions and execution rollbacks. It has good performance when all federates can advance their simulation time with comparable speed [6]. This usually requires that federates should have homogeneous simulation workload and should execute on homogeneous computational resources. Unfortunately, it is very difficult to meet such requirement in practice. Federates which simulate different parts of an unsymmetric system have probably different workloads. Furthermore, the workload of each federate might change during the simulation execution. In the traditional execution platform, federates might execute on heterogenous computing nodes. In VEE, federates may share resources with other federates or even other applications.

In this paper, each federate is encapsulated and executed on a *virtual machine* (VM). VMs consolidated on a computing node are scheduled by a hypervisor to share the underlying resources. By setting relevant parameters of the

VM scheduling algorithm, VMs can be constrained to use a desired share of computational resources, i.e., CPU compute cycles (to be explained in Section 3.2). Consequently, the performance of federates can be controlled. Inspired to this, we propose an adaptive resource provisioning system for the purpose of accelerating optimistic HLA-based simulations in VEE. A performance monitor is introduced using a middleware approach to measure the execution performance of federates transparently to the simulation application. A resource manager is developed as a bridge between the performance monitor and hypervisor. It periodically retrieves performance measurements of federates from the performance monitor and the amount of available resources from the hypervisor. Based on the performance measurements, the resource manager can predict workload of the federates and then distribute the available resources to the federates accordingly. As a result, federates are able to advance their simulation time with comparable speed, avoiding over-optimistic executions and execution rollbacks. Hence, the simulation execution can be accelerated using the same amount of computational resources.

The rest of the paper is organized as follows: Section 2 discusses the related work. Section 3 introduces some background knowledge including HLA-based simulations and VM scheduling in VEE. Section 4 illustrates the details of our proposed adaptive resource provisioning system. Section 5 describes the experiment design and discusses the experimental results. Finally, Section 6 concludes the paper and outlines the future work.

2. RELATED WORK

A number of research work have been done on accelerating parallel and distributed simulations on traditional execution platforms. Federate migration protocols [15] were proposed to achieve workload balance in the federation execution. Different synchronization approaches are mixed [25] or replicated [16] for the purpose of taking their advantages in the same simulation execution. The *Adaptive Time Warp* protocol [24] improves performance of optimistic synchronization by reactively blocking the optimistic execution of simulation components. Specifically, a simulation component is blocked if it has executed far beyond others in the same simulation execution. The protocol is most closely related to our proposed approach. However, its performance is sensitive to the frequency of global synchronization, which is time-consuming in the Cloud due to the unstable network performance [30].

Recently, researchers have tried to tackle the technical challenges that arise in moving parallel and distributed simulations to the Cloud [26, 6]. Malik et. al [17] have proposed a novel protocol to enhance optimistic synchronization efficiency on the Cloud in the presence of high network latency and jitters. Similar to the *Adaptive Time Warp* protocol [24], the protocol proposed in [17] reactively blocking the execution of fast simulation components. In contrast, we propose to proactively make federates advance simulation time with comparable speed, and thus, avoids execution rollbacks.

In addition, researchers have also tried to accelerate parallel and distributed simulations on modern many-core processors [7, 4]. Similar to our proposed approach, Child and Wilsey [4] have explored the features of Dynamic Voltage and Frequency Scaling (DVFS) for the purpose of reducing the execution rollbacks in optimistic simulations. The performance of individual simulation components are controlled by adjusting the frequency and voltage of their corresponding CPU cores. However, due to the hardware limit, only four to six discrete frequency-voltage pairs are made possible. To the best of our knowledge, we are the first to consider accelerating HLA-based simulations by harnessing virtualization technologies which are commonly used in the Cloud. Different from DVFS, the virtualization technologies leveraged in this paper enable fine-grained adjustment of CPU resources (to be explained in Section 3.2).

In Cloud computing, under-provisioning resources will cause *Service Level Objective* (SLO) violation, resulting in financial penalties; while over-provisioning resources will increase cost and waste resources that could be assigned to other users. To avoid these problems, several elastic resource allocation systems [10, 28, 14, 22] have been proposed. In [10, 28], signature-driven and state-driven approaches are used to predict application workload. According to the workload prediction, resources are automatically allocated to the corresponding VMs. In [14, 22], the amount of resources necessary to achieve application SLO is automatically determined using a control theory-based estimator. Unfortunately, these resource allocation systems are targeted at web server applications and cannot be directly applied on optimistic HLA-based simulations. For instance, the workload of web server applications can be predicted according to historical CPU utilization rate [10, 28], as the CPU is put to idle when the workload is light. In contrast, optimistic federates consume allocated CPU resource on optimistic executions which might be rolled back in the future. Hence, the CPU utilization rate does not represent the useful workload of federates. In addition, the performance target of server applications are well defined in SLO and their real performance can be measured easily [14, 22]. In contrast, the performance of an HLA-based simulation is concerned with the performance of individual federates, as well as the time synchronization among them. It is also non-trivial to measure federate performance transparently to the simulation applications.

3. BACKGROUND

3.1 HLA-based Simulations

In an HLA-based simulation, federates might be developed by different participants from different organizations. They are executed in a parallel and distributed manner. The HLA interface specification implemented by a *Runtime Infrastructure* (RTI) defines how a federate interacts with other federates in the federation. Usually, an RTI component is provided by the RTI to connect each joined federate to the federation. It keeps part of the RTI state concerned with the federate and takes care of the federate's need to exchange messages and to request time advancement. The bidirectional interface between the federate and its RTI component is usually implemented via ambassadors. Federates invoke services provided by the RTI ambassador, while the RTI delivers callbacks to the federate through its federate ambassador. For instance, a federate sends messages by invoking the corresponding RTI services (e.g., *SendInteraction*). These messages are delivered to receiving federates in the form of callbacks (e. g., *ReceiveInteraction*).

An optimistic federate invokes *Flush Queue Request* (FQR)

service iteratively during the simulation execution. It forces the RTI to deliver all buffered messages to the federate. In the mean time, the FQR request might trigger a federation-wide time synchronization. The time granted by the RTI after FQR service is referred to as *logical time* of the federate. It defines the lower bound of future execution rollbacks. It can be used for fossil collection in the federate, e.g., the storage space containing checkpointed states with TS smaller than the granted time can be reclaimed. Since the federate is allowed to process events freely, its *simulation time* (i.e., TS of the event being processed) might be greater than its logical time. However, it might receive a straggler message whose TS is smaller than its simulation time. If so, a causality error happens and the federate needs to rollback its execution by discarding the *over-optimistic execution* (i.e., the execution of those events with TS greater than the TS of the straggler message). It must undo the modification on federate state relying on federate state saving and restoration mechanism [13, 9]. Additionally, it needs to invoke *Retract* services to unsend those incorrect messages which were sent during the over-optimistic execution. In the case that the messages have already been delivered to the receiving federates, the RTI informs those federates to remove the effect of the incorrect messages through *Request Retraction* (RR) callbacks. This may cause secondary execution rollbacks in the receiving federates.

3.2 VM Scheduling

In VEE, the hypervisor monitors the underlying resources and performs VM scheduling. Credit scheduler [33], the default CPU scheduler in Xen, is a proportional share scheduler. The credit allocation for VMs is managed by two parameters: a weight and an optional cap. By default, only weight is defined for each VM. In this case, the credit scheduler is work-conserving, which means that a VM that has expended all of its credit will be allocated additional CPU share if no other VM is using its allocated share. Optionally, we can use cap to specify the maximum CPU share the VM is allowed to consume. In this case, the credit scheduler is non-work-conserving, which means that a VM never consumes CPU share beyond the cap even if the rest of CPU share is left idle. Compared with work-conserving, non-work-conserving provides better performance isolation among VMs [3, 27]. The cap value, rather than the weight value, precisely specifies the CPU share allocated to a VM. It can be any integer value (denoted as c) between 1 and 100, indicating that c% time of one CPU core is scheduled for the VM [1]. In this paper, the computational resources allocated to federates are controlled through the fine-grained adjustment of the CPU caps of their resident VMs.

4. ADAPTIVE RESOURCE PROVISIONING SYSTEM

4.1 Architecture Overview

The architecture of our proposed adaptive resource provisioning system is illustrated in Figure 1. Each federate is executed on its individual resident VM. Similar to the solutions in [31], a performance monitor can be inserted between

[1]The cap value can also be greater than 100, indicating that more the one CPU core is scheduled for the VM

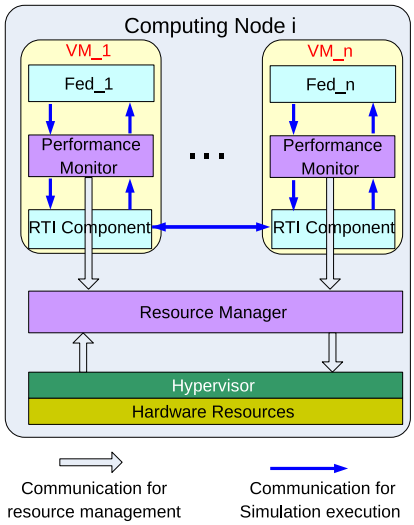


Figure 1: Overview of the adaptive resource provisioning system

the federate and its RTI component as a middleware. By intercepting the RTI services and callbacks, the performance monitor is able to measure the performance of the federate transparently to the simulation application. More details will be provided in Section 4.2.

A resource manager is developed as a bridge between the performance monitor and the hypervisor of the computing node. It periodically adjusts the resource shares for federates in the federation for the purpose of making them have the same performance. The simulation execution is divided into a number of successive control intervals. The control interval length is a parameter used in the adaptive resource provisioning system. In each control interval, the resource manager retrieves performance of federates through their performance monitors. In the meantime, it retrieves the available resources of the computing node through its hypervisor. An algorithm is developed to distribute the available resources among federates. Finally, the resource manager sends resource shares of federates (i.e., caps of their resident VMs) to the hypervisor, and the latter schedules VMs accordingly (as described in Section 3.2). By allocating appropriate computational resource shares, the federates with different workloads are able to achieve comparable performance. Compared with the scheme that evenly distributes computational sources among federates, the adaptive resource provisioning system allocates less computational resources to federates with low workload. Hence, they can avoid over-optimistic executions and execution rollbacks. On the other hand, the adaptive resource provisioning system allocates more computational resources to federates with high workload. Hence, the overall federation execution can be accelerated. More details will be provided in Section 4.3.

Although only one computing node is illustrated in Figure 1, the adaptive resource provisioning system can be easily extended to multiple computing nodes for large-scale HLA-based simulations. In this case, the resource managers deployed at different computing nodes should coordinate with each other for the purpose of making all federates in

the federation have the same performance. In addition, the resident VMs should be migrated among computing nodes if necessary to achieve workload balance and/or to reduce communication overhead [5].

4.2 Performance Monitor

In HLA-based simulations, the performance of a federate is measured by its execution speed, i.e., how fast the federate can advance its simulation time. Intuitively, it can be measured as the advanced simulation time in a control interval divided by the control interval length. However, execution rollbacks might happen during the control interval. If so, the simulation time decreases and the federate wastes time on over-optimistic executions and execution rollbacks. Hence, the execution speed measured in this manner may not accurately characterize the useful workload of the federate. In some extreme cases, the measured execution speed might even be a negative value. To solve this problem, the simulation execution is partitioned into epochs, which is usually much smaller than the control interval. Only the epochs without over-optimistic executions and execution rollbacks are taken into account for performance measurement.

As shown in Figure 1, the performance monitor in the middleware is able to intercept FQR services invoked by federates. By default, the argument (i.e., the *requested time*) of a FQR service is the TS of the next internal event in the federate. Usually, it is quite close to the simulation time of the federate. Hence, we can use the requested time to estimate the progress of federates approximately[2]. In this way, federate execution can be partitioned into epochs, each of which is defined as the requested time interval between two subsequent FQR requests. Usually, federates are not required to invoke a FQR service before processing every event. Hence, an epoch is likely to include multiple processed events.

Algorithm 1 Intercept FQR service

```
1: Let E be the list of epochs in each control interval
2: if RollbackFlag == true then
3:     RollbackFlag = false
4: else
5:     Append a new epoch (E_k) at the end of E
6:     E_k → begTS = ReqTime(k − 1)
7:     E_k → endTS = ReqTime(k)
8:     E_k → advTS = ReqTime(k) − ReqTime(k − 1)
9:     E_k → exeTime = Clock(k) − Clock(k − 1)
10: end if
```

As shown in Algorithm 1, the epochs in the control interval are kept in a list denoted as **E**. On intercepting the k^{th} FQR invocation, the performance monitor creates a new epoch E_k (Lines 5), except that an execution rollback is detected (to be discussed later) before the FQR invocation (Lines 2 and 3). In the meantime, it also records the requested time denoted as $ReqTime(k)$ and the wall time clock denoted as $Clock(k)$. Then, $ReqTime(k-1)$ and $ReqTime(k)$ respectively defines the beginning and the end of E_k (Lines 6 and 7). The advanced simulation time and execution time of E_k is $ReqTime(k) - ReqTime(k-1)$ and $Clock(k) - Clock(k-1)$ respectively (Lines 8 and 9).

As mentioned in Section 3.1, messages and RR callbacks are delivered to federates after each FQR invocation. In order to detect the execution rollbacks, the performance monitor intercepts the delivered messages and RR callbacks, as shown in Algorithm 2. Firstly, it calculates $minMsgTS$, i.e., the minimum TS of those messages and RR callbacks (Line 2). In the case that the $minMsgTS$ is smaller than the $endTS$ of an epoch in **E**, the epoch includes over-optimistic execution which will be rolled back. Therefore, it should not be considered for performance measurement (Lines 3 to 5). Once an execution rollback is detected, the $rollbackFlag$ is set to true. This will prevent the performance monitor from creating an epoch on intercepting the subsequent FQR invocation (Lines 2 and 3 in Algorithm 1).

Algorithm 2 Intercept messages and RR callbacks

```
1: Let M be the list of delivered messages and RR callbacks
   after each FQR service
2: minMsgTS = min_{M_i ∈ M} M_i → TS
3: for Each E_i ∈ E do
4:     if minMsgTS < E_i → endTS then
5:         Remove E_i from E
6:         RollbackFlag = true
7:     end if
8: end for
```

At the end of each control interval, the performance monitor is required to measure the execution speed of the federate during the control interval, as shown in Algorithm 3. It is possible that the control interval might not include any epoch, especially when the control interval is too short and/or when the federate encounters an execution rollback. In this case, the performance monitor simply reports an invalid execution speed, i.e., NaN (Not a Number). Otherwise, the execution speed of the federate is calculated as the sum of advanced simulation time of all epochs in the control interval divided by the sum of execution time of those epochs. After the performance is measured, the list **E** is able to be cleared up for keeping epochs in the subsequent control interval.

Algorithm 3 Measure performance

```
1: if E is empty then
2:     exeSpeed = NaN //NaN: Not a Number
3: else
4:     exeSpeed = (Σ_{E_i ∈ E} E_i → advTS) / (Σ_{E_i ∈ E} E_i → exeTime)
5: end if
6: Remove all elements from E
7: return exeSpeed
```

4.3 Resource Manager

Since the workload prediction is not the main focus of this paper, we simply use the simulation workload in the current control interval to predict that in the subsequent control interval. It is also assumed that federate performance increases proportionally with increasing VM Cap [27]. Take the i^{th} federate as an example, at the end of k^{th} control interval, the resource manager retrieves the *measured execution speed* denoted as $MES_i(k)$ from the performance monitor.

[2]For better accuracy, the performance monitor can provide federate with a modified FQR interface including both simulation time and requested time arguments. However, such modified interface is not compliant to the HLA standard.

Suppose that the caps of its resident VM at k^{th} and $(k+1)^{th}$ control intervals are set to $Cap_i(k)$ and $Cap_i(k+1)$ respectively, we can get the *predicted execution speed* (PES) at the $(k+1)^{th}$ control interval as:

$$PES_i(k+1) = \frac{MES_i(k)}{Cap_i(k)} \times Cap_i(k+1) \quad (1)$$

To accelerate the simulation execution, the PES of all federates should be maximized at every control interval. However, some constraints should be satisfied in the meantime. Firstly, the federates can only consume the available resource. Suppose that the federation scale is N; the simulation execution time is T; the control interval length is t; the number of available CPU cores is M, we can get

$$\sum_{i=1}^{N} Cap_i(k) \leq M * 100 \quad (2)$$

for each $k \in \{1, 2, ..., \frac{T}{t}\}$. Secondly, each federate cannot consume more than one CPU core, because the federate, as a simulation component, is usually single-threaded [18]. That is,

$$Cap_i(k) \leq 100 \quad (3)$$

for each $i \in \{1, 2, ..., N\}$ and $k \in \{1, 2, ..., \frac{T}{t}\}$. Thirdly, to avoid execution rollbacks, federates should have the same PES in the same control interval. That is,

$$PES_i(k) = PES_j(k) \quad (4)$$

where i and $j \in \{1, 2, ..., N\}$ and $k \in \{1, 2, ..., \frac{T}{t}\}$.

The functionality of the resource manager is illustrated in Algorithm 4. $\mathbf{F}$ denotes the set of all federates in the federation. At the end of each control interval, the resource manager retrieves MES from the performance monitor of all federates in the federation (Lines 5 to 10). In the case that the retrieved MES is NaN (Not a Number), the MES is approximately set as the PES of the federate in the control interval. After that, the resource manager distributes available resources to federates in the federation for the purpose of maximizing PES of all federates while satisfying aforementioned constraints.

According to Equation 1, the federate with smaller $\frac{MES(k)}{Cap(k)}$ should have greater $Cap(k+1)$ to meet the third constraint (Equation 4) in the next control interval. In other words, F_m with the minimum $\frac{MES(k)}{Cap(k)}$ should have the maximum $Cap(k+1)$ in the federation. Due to the second constraint (Equation 3), $Cap_m(k+1)$ can be set to 100 at the most. Without considering the first constraint (Equation 2), we can initially set $Cap_m(k+1) = 100$. Then, $PES_m(k+1)$ can be calculated according to Equation 1 (Line 13). According to the third constraint, other federates should have the same PES as F_m (Line 15). Then, $Cap(k+1)$ for other federates can be calculated by rewriting Equation 1 (Line 16).

However, the first constraint (Equation 2) will not be satisfied if the ratio of $M \times 100$ to $\sum_{F_i \in \mathbf{F}} Cap_i(k+1)$, denoted as φ, is smaller than one. To make it satisfied, the caps of all federates should be reduced by φ times. Obviously, the second constraint will be satisfied as well. Since the PES of all federates will be reduced by φ times, they will remain the same, making the third constraint satisfied. In the case that $\varphi > 1$, the federates in the federation cannot consume all the available CPU resources. F_m becomes the bottleneck that holds back the simulation execution.

Algorithm 4 Resource Manager

```
1: F = {F_1, F_2, ..., F_N};
2: k = 0;
3: while k < T/t do
4:    Thread.sleep(t);
5:    for each F_i ∈ F do
6:       MES_i(k) = F_i → measurePerformance();
7:       if MES_i(k) == NaN then
8:          MES_i(k) = PES_i(k);
9:       end if
10:   end for
11:   Choose F_m ∈ F where MES_m(k)/Cap_m(k) = min_{F_i∈F} MES_i(k)/Cap_i(k);
12:   Cap_m(k+1) = 100;
13:   PES_m(k+1) = MES_m(k)/Cap_m(k) × Cap_m(k+1);
14:   for each F_i ∈ F − {F_m} do
15:      PES_i(k+1) = PES_m(k+1);
16:      Cap_i(k+1) = PES_i(k+1)/MES_i(k) × Cap_i(k) ;
17:   end for
18:   φ = M×100 / Σ_{F_i∈F} Cap_i(k+1);
19:   if φ < 1 then
20:      for each F_i ∈ F do
21:         Cap_i(k+1) = φ × Cap_i(k+1);
22:         PES_i(k+1) = φ × PES_i(k+1);
23:      end for
24:   end if
25:   k++;
26: end while
```

5. EXPERIMENTS AND RESULTS

5.1 Experiment Design

A *Massively Multiplayer Online Games* (MMOGs) ecosystem simulation model is used in our experiments to evaluate our proposed adaptive resource provisioning system. Nae et. al [21] have proposed a dynamic resource provisioning method and introduced the concepts of MMOGs ecosystem. In the MMOGs ecosystem, game operators rent resources (CPU, Network and Memory) from data centers for running the MMOG servers. They are able to dynamically adjust the amount of renting resources according to the workload of MMOGs (e.g., the number of players and their interactions). We develop an HLA-based simulation to simulate the MMOGs ecosystem. It can be used to study the effect of dynamic resource provisioning schemes using different workload prediction algorithms and different resource hosting policies.

In our HLA-based simulation, each federate simulates the game servers in a data center. It generates a local event to simulate the performance and resource provisioning of the game servers in each time step (i.e., 10 seconds) of the MMOGs execution [21]. That is, the difference of simulation time between any two successive local event is 10. Each local event calculates the response time of each interaction initialized by players connected to the data center and measures the quality of game experience from the perspective of those players, using the resources rented from the data

center in the corresponding time step. Therefore, the simulation model can be used to evaluate resource provisioning schemes using performance metrics such as resource over-allocation and resource under-allocation [21]. In addition, we can also get that the computational resource required for processing a local event increases proportionally with the increasing number of players connected to the data center in the corresponding time step. In our experiments, the number of players connected to the data centers are retrieved from the trace of RuneScape [21]. They have strong diurnal pattern. Therefore, the simulation workload of the corresponding federates also have strong diurnal pattern. In the case that the data centers locate at regions in different time zones, the corresponding federates may have different simulation workloads at the same simulation time. For simplicity, the federation scale is set to two in our experiments. Various cases of simulation workloads (specifically, the number of players connected to the corresponding data centers) are introduced in the two federates. In Case 1, the data centers locate in the regions within the same time zone. In Cases 2 and 3, the time difference between the data centers is around 6 and 12 hours, respectively. For instance, the number of players during a day in Case 3 is illustrated in Figure 2.

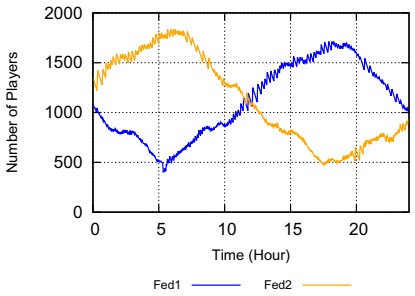


Figure 2: Simulation workload of federate (Case3)

In addition, federates might send external events to each other to simulate the communication between servers located at different data centers. For instance, servers may replicate some parts of their game state in different data centers to tolerate unpredictable failures [19]. Depending on different strategies, state replications might be triggered at different situations and the replication frequency might vary in a large range. For simplicity, we assume each external event is triggered by an internal event. The probability of generating external events is denoted as $P_{ExternEvent}$. The TS of an external event equal to TS of the processing internal event plus a *lookahead* (LA) [9], indicating that the external event will affect the receiving federate after LA simulation time. $P_{ExternEvent}$ and LA are used as parameters in our HLA-based simulation. In our experiments, we will evaluate the performance of a number of simulation executions with different parameter inputs, i.e., different values of $P_{ExternEvent}$ and LA.

Optimistic synchronization is employed in the MMOGs ecosystem simulation model. Federates are allowed to process three events before invoking the next FQR service. In addition, the infrequent state saving approach described in [8] is adopted. Federates save their state after processing 15 events. When a causality error happens, the federate must roll back to a previous saved state and coast-forward execution to recreate the desired execution state.

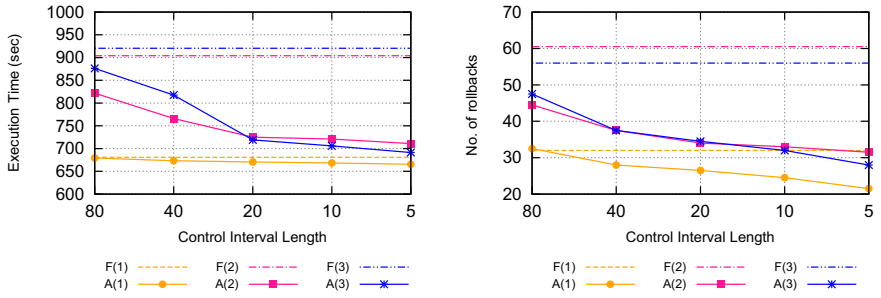
The simulation executes on an RTI implemented by reusing the code in SOHR [23]. For efficiency consideration, RTI components are implemented as the libraries of their corresponding federates; the communication among RTI components use JAVA socket instead of the heavy Grid service invocation. Experiments are carried out on a computing node installed with 12 Intel Xeon 2.67GHz CPU cores, 24 GB RAM, CentOS 6.2 and Xen 4.1.2. Each federate executes on a VM with one VCPU core, 2 GB RAM and CentOS 6.2.

Two execution scenarios are studied. In the *Fixed* scenario, the caps of VMs are fixed at 50, i.e., two federates share one CPU core evenly. In the *Adaptive* scenario, the caps of the resident VMs are dynamically adjusted using our proposed adaptive resource provisioning system. However, the sum of their caps is always equal to 100. The simulation length is 2 days, including the first day as the warm up period.

5.2 Experiment Results

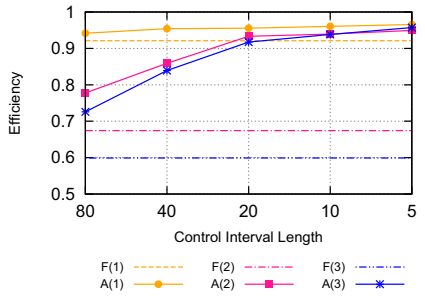
In the first series of experiments, we compare the simulation execution performance in *Fixed* and *Adaptive* scenarios with aforementioned workload cases. The experimental results are illustrated in Figure 3, including the simulation execution time, the number of execution rollbacks and the execution efficiency. The execution efficiency is defined as the ratio of useful events to total events processed [17]. Besides the useful events, federate might process events in the over-optimistic execution and the coast-forward execution. The *Fixed* scenario (*Adaptive* scenario) for workload cases 1, 2 and 3 are respectively denoted as $F(1)$, $F(2)$ and $F(3)$ ($A(1)$, $A(2)$ and $A(3)$) in the figure. The control interval length as the parameter in the adaptive resource provisioning system ranges from 80 to 5 seconds. Obviously, it is meaningful for the *Adaptive* scenario, but not the *Fixed* scenario. For simplicity, the parameters of the simulation model are set as follows: $P_{ExternEvent} = 1\%$ and $LA = 45$. Different parameter inputs of the simulation model will be studied latter.

In Case 1, federates have similar workload during the simulation execution. Therefore, federates executed in both *Fixed* and *Adaptive* scenarios are able to advance simulation time with comparable speed. Hence, federates encounters a small number of execution rollbacks (Figure 3(b)); and they have high execution efficiency (Figure 3(c)). Therefore, we can observe from Figure 3(a) that the *Fixed* and *Adaptive* scenarios have similar execution performance. In Cases 2 and 3, federates have different workload. For the *Fixed* scenario, compared with Case 1, the numbers of execution rollbacks are much greater (Figure 3(b)). Furthermore, the over-optimistic execution discarded in each execution rollback can be very huge due to the different execution speed of federates. As a result, the execution efficiency are much lower (Figure 3(c)). Therefore, we can observe from Figure 3(a) that the execution time of $F(2)$ and $F(3)$ are much greater than that of $F(1)$. Different from the *Fixed* scenario, federates in the *Adaptive* scenario have comparable execution speed. They encounter less execution rollbacks (Figure 3(b)) and have higher execution efficiency (Figure 3(c)). Therefore, our proposed adaptive resource provisioning system is able to reduce the simulation execution time significantly (Figure 3(a)).



(a) Execution Time

(b) Execution Rollbacks



(c) Execution Efficiency

Figure 3: Performance comparison with different workload cases and decreasing control interval length

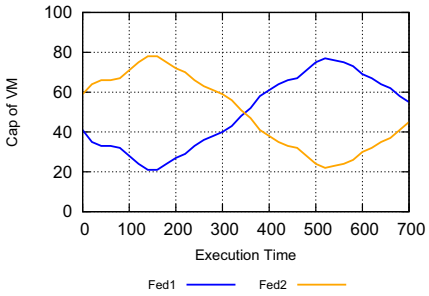
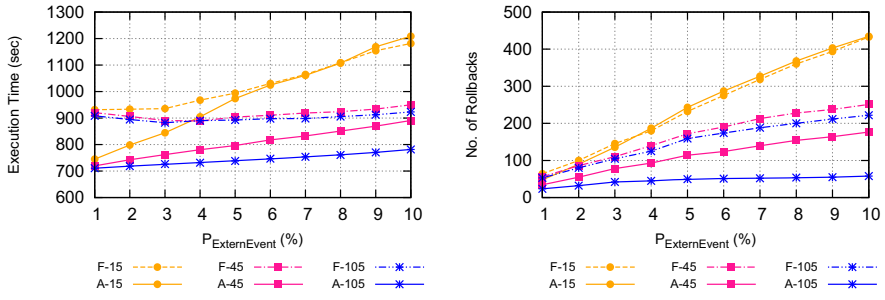


Figure 4: Caps of the resident VMs (Case3)

Generally speaking, the smaller the control interval length, the more accurately the resource manager is able to adjust the resource provisioning according to simulation workload. As a result, federates encounter less execution rollbacks (Figure 3(b)) and have higher execution efficiency (Figure 3(c)). Therefore, we can observe from Figure 3(a) that the simulation execution time in *Adaptive* scenario decreases with the decreasing control interval length. When the control interval length is equal to 5 seconds, the adaptive resource provisioning system can obtain 22% and 25% *performance enhancement* (i.e., the difference in execution time between *Fixed* and *Adaptive* scenarios divided by the execution time in *Fixed* scenario) for Cases 2 and 3 respectively. It is worth to point out that federates executed in the *Adaptive* scenario cannot avoid all execution rollbacks. This is because that they cannot advance their simulation time with the same speed in practice, due to the accuracy of workload prediction and performance overhead in VEE [20]. Figure 4 illustrates the caps of resident VMs of federates during the simulation execution when the control interval length is 20 seconds. By comparing Figure 2, we can observe that the CPU resource is proportionally distributed to federates according to their simulation workloads.

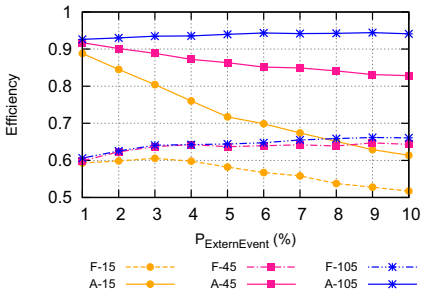
The second series of experiments are carried out to compare the simulation execution performance in *Fixed* and *Adaptive* scenarios with different parameter inputs of the simulation model. Due to the space limit, we only report the experimental results for Case 3 simulation workload, which is quite common in the MMOGs ecosystem, as the data centers usually locate at different places around the world. In addition, the control interval length is set to 20 seconds. Figure 5 illustrates the simulation execution, number of execution rollbacks and execution efficiency with $P_{ExternEvent}$ increasing from 1% to 10%. Three different LA values (i.e., 15, 45 and 105) are taken into account. The *Adaptive* and *Fixed* scenarios for $LA = l$ are denoted as $A - l$ and $F - l$ respectively.

As aforementioned, our proposed adaptive resource provisioning system does not ensure that federates advance their simulation time with the same speed in practice. Hence, we can observe from Figure 5(b) that, federates in the *Fixed* and *Adaptive* scenarios encounter similar number of execution rollbacks when LA is too small (e.g., $LA = 15$). In the case that $P_{ExternEvent}$ is small, the over-optimistic execution discarded in each execution rollback is much longer in the *Fixed* scenario than that in the *Adaptive* scenario. For this reason, the *Adaptive* scenario has much higher execution efficiency (Figure 5(c)) and outperforms the *Fixed* scenario significantly (Figure 5(a)). With the increasing $P_{ExternEvent}$, the over-optimistic execution decreases as the faster federate is frequently rolled back by the slower federate due to the straggler messages. As a result, the differ-



(a) Execution Time

(b) Execution Rollbacks



(c) Execution Efficiency

Figure 5: Performance comparison with different LA and increasing $P_{ExternEvent}$

ence in the execution efficiency and execution time between *Adaptive* and *Fixed* scenarios decreases.

When LA is large (e.g., $LA = 105$), federates in the *Adaptive* scenario with comparable execution speed are able to avoid almost all rollbacks (Figure 5(b)). As a result, the execution efficiency in *Adaptive* scenario is close to one and is much higher than that in the *Fixed* scenario. Furthermore, the number of execution rollbacks and execution efficiency are almost the same regardless of the increasing $P_{ExternEvent}$. Therefore, we can observe from Figure 5(a) that the adaptive resource provisioning system can always achieve significant performance enhancement. When $LA = 45$, the *Adaptive* scenario can still outperform the *Fixed* scenario. However, the performance enhancement significantly decreases with the increasing $P_{ExternEvent}$, as the number of execution rollbacks increases and the execution efficiency decreases in the *Adaptive* scenario.

Figure 6 illustrates the performance enhancement of the adaptive resource provisioning system with respect to various parameter inputs of the simulation model. As we can see, the adaptive resource provisioning system can achieve significant performance enhancement except for those parameter inputs $LA = 15$ and $P_{ExternEvent} > 6\%$. This is because that the number of execution rollbacks and over-optimistic executions discarded in execution rollbacks are similar in *Adaptive* and *Fixed* scenarios. With the decreasing $P_{ExternEvent}$, the performance enhancement increases as the over-optimistic execution in the *Fixed* scenario increases. With the increasing LA, the performance enhancement increases as the number of execution rollbacks in the *Adaptive* scenario decreases.

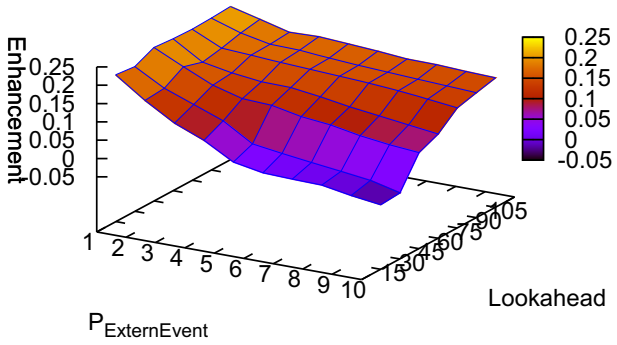


Figure 6: Performance enhancement of the adaptive resource provisioning system

6. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an adaptive resource provisioning system for the purpose of accelerating optimistic HLA-based simulations in VEE. Using a middleware approach, the performance monitor is able to measure the performance of individual federates transparently. Based on the performance measurements, the resource manager can predict the workload of federates, and then, adjust their resource allocations accordingly by changing the resource shares of their VMs. In this way, federates in the same federation can advance their simulation time with comparable speeds, and thus, avoid wasting resources on over-optimistic executions and execution rollbacks. Experiments are carried

out using a MMOGs ecosystem simulation model where federates are likely to have dynamic and imbalanced simulation workload. Furthermore, different parameter inputs of the simulation model are investigated. As experimental results have shown, the adaptive resource provisioning system can significantly accelerate the simulation executions with most of the investigated parameter inputs.

For simplicity, we have assumed that federates are single-threaded in this paper. However, in order to exploit the computational potential of multi-core processors, a federate can be a group of concurrent and tightly dependent threads [12]. In the future, we will extend our proposed adaptive resource provisioning system to support the executions of multi-threaded federates. Their corresponding VMs will have multiple virtual CPU cores and the cap values can be greater than 100. In addition, co-scheduling solutions [32, 29] where virtual CPU cores of the same VM are scheduled simultaneously will be adopted to reduce the synchronization latency among the threads in the same federate. In the meantime, we will also evaluate the scalability of our system using large scale simulations on large execution environments with many computing nodes. With the increasing simulation scale, the communication and computation overhead of our system increase, as shown in algorithm 4. However, on the other hand, the problem of workload imbalance become more critical. As for future work, we also plan to extend our proposed adaptive resource provisioning system by adopting more accurate workload prediction algorithms [10, 14, 22, 28]. Besides HLA-based simulations, our system will also be applied and integrated to other parallel and distributed applications.

7. ACKNOWLEDGMENTS

This work is supported under future data center technology thematic strategic research programme by Singapore Agency for Science, Technology and Research (A*STAR) with grant number 112 172 0015.

8. REFERENCES

[1] Amazon. Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, Oct. 2003.

[3] S. K. Barker and P. Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Procs of conference on Multimedia systems (MMSys'10)*, pages 35–46, 2010.

[4] R. Child and P. A. Wilsey. Using DVFS to optimize time warp simulations. In *Procs of the 44th Conference on Winter Simulation (WSC '12)*, 2012.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Procs of conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*, pages 273–286, 2005.

[6] G. D'Angelo. Parallel and distributed simulation from many cores to the public cloud. In *Procs of the International Conference on High Performance Computing and Simulation (HPCS'11)*, pages 14–23, 2011.

[7] G. D'Angelo, S. Ferretti, and M. Marzolla. Time warp on the go. In *Procs of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 242–248, 2012.

[8] J. Fleischmann and P. A. Wilsey. Comparative analysis of periodic state saving techniques in time warp simulators. In *Procs of workshop on Parallel and distributed simulation(PADS '95)*, pages 50–58, 1995.

[9] R. M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley Interscience, 2000.

[10] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Procs of International Conference on Network and Service Management (CNSM'10)*, pages 9–16, 2010.

[11] IEEE. *1516-2010 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules*, August 2010.

[12] D. Jagtap, N. Abu-Ghazaleh, and D. Ponomarev. Optimization of parallel discrete event simulator for multi-core systems. In *Procs of the 26th International Parallel and Distributed Processing Symposium (IPDPS'12)*, pages 520–531, 2012.

[13] D. R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and System*, 7(3):404–425, 1985.

[14] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Procs of international conference on Autonomic computing (ICAC'09)*, pages 117–126, 2009.

[15] Z. Li, W. Cai, S. J. Turner, and K. Pan. Federate migration in a service oriented HLA RTI. *Procs of Symposium on Distributed Simulation and Real-Time Applications (DS-RT '07)*, pages 113–121, 2007.

[16] Z. Li, W. Cai, S. J. Turner, and K. Pan. Improving performance by replicating simulations with alternative synchronization approaches. In *Procs of the 40th Conference on Winter Simulation (WSC '08)*, pages 1112–1120, 2008.

[17] A. Malik, A. Park, and R. Fujimoto. Optimistic synchronization of parallel simulations in cloud computing environments. In *Procs of the Conference on Cloud Computing (CLOUD'09)*, pages 49–56, 2009.

[18] D. E. Martin, T. J. McBrayer, and P. A. Wilsey. Warped: A time warp simulation kernel for analysis and application development. In *Procs of the 29th Hawaii International Conference on System Sciences Volume 1: Software Technology and Architecture (HICSS'96)*, pages 383–386, 1996.

[19] J. Mason. A detailed look at data replication options for disaster recovery planning. White Paper, June 2009.

[20] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *Procs of the 1st ACM/USENIX international conference on Virtual execution environments (VEE'05)*, pages 13–23, 2005.

[21] V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D. Epema,

and T. Fahringer. Efficient management of data center resources for massively multiplayer online games. In *Procs of the conference on Supercomputing (SC'08)*, pages 10:1–10:12, 2008.

[22] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Procs of European conference on Computer systems (EuroSys'09)*, pages 13–26, 2009.

[23] K. Pan, S. J. Turner, W. Cai, and Z. Li. A service oriented HLA RTI on the Grid. In *Procs of Conference on Web Services (ICWS 2007)*, pages 984–992, 2007.

[24] K. S. Panesar and R. M. Fujimoto. Adaptive flow control in time warp. In *Procs of workshop on Parallel and distributed simulation (PADS '97)*, pages 108–115, 1997.

[25] K. S. Perumalla. μsik - a micro-kernel for parallel/distributed simulation systems. In *Procs of Workshop on Principles of Advanced and Distributed Simulation (PADS '05)*, pages 59–68, 2005.

[26] A. J. P. R. M. Fujimoto, A. W. Malik. Parallel and distributed simulation in the Cloud. *SCS Modeling and Simulation Magazine, Society for Modeling and Simulation, Intl.*, 1, July 2010.

[27] D. Schanzenbach and H. Casanova. Accuracy and responsiveness of cpu sharing using xenâĂŹs cap values. Technical report, 2008.

[28] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Procs of Symposium on Cloud Computing (SOCC'11)*, pages 5:1–5:14, 2011.

[29] O. Sukwong and H. S. Kim. Is co-scheduling too expensive for smp vms? In *Procs of the sixth conference on Computer systems (EuroSys'11)*, pages 257–272, 2011.

[30] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *Procs of conference on Information communications (INFOCOM'10)*, pages 1163–1171, 2010.

[31] X. Wang, S. J. Turner, M. Y. H. Low, and B. P. Gan. Optimistic synchronization in HLA-based distributed simulation. *Simulation*, 81:279–291, April 2005.

[32] C. Weng, Q. Liu, L. Yu, and M. Li. Dynamic adaptive scheduling for virtual machines. In *Procs of the 20th international symposium on High performance distributed computing (HPDC'11)*, pages 239–250, 2011.

[33] Xen. Xen Credit Scheduler. http://wiki.xen.org/wiki/Credit _Scheduler.