Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

1-2015

Adaptive resource provisioning mechanism in VEEs for improving performance of HLA-based simulations

Zengxiang LI

Wentong CAI

Stephen John TURNER

Xiaorong LI

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

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the Software Engineering Commons

Citation

LI, Zengxiang; CAI, Wentong; TURNER, Stephen John; LI, Xiaorong; and TA, Nguyen Binh Duong. Adaptive resource provisioning mechanism in VEEs for improving performance of HLA-based simulations. (2015). *ACM Transactions on Modeling and Computer Simulation*. 26, (1), 1:1-1:25. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/4848

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Adaptive Resource Provisioning Mechanism in VEEs for Improving Performance of HLA-Based Simulations

ZENGXIANG LI, Institute of High Performance Computing, Singapore
WENTONG CAI and STEPHEN JOHN TURNER, Nanyang Technological University, Singapore
XIAORONG LI, Infocomm Development Authority of Singapore, Singapore
TA NGUYEN BINH DUONG, Ngee Ann Polytechnic, Singapore
RICK SIOW MONG GOH, Institute of High Performance Computing, Singapore

Parallel and distributed simulations (or *High-Level Architecture* (HLA)-based simulations) employing optimistic synchronization allow federates to advance simulation time freely at the risk of overoptimistic executions and execution rollbacks. As a result, the simulation performance may degrade significantly due to the simulation workload imbalance among federates. In this article, we investigate the execution of parallel and distributed simulations on Cloud and data centers with *Virtual Execution Environments* (VEEs). In order to speed up simulation execution, an *Adaptive Resource Provisioning Mechanism in Virtual Execution Environments* (ArmVee) is proposed. It is composed of a performance monitor and a resource manager. The former measures federate performance transparently to the simulation application. The latter distributes available resources among federates based on the measured federate performance. Federates with different simulation workloads are thus able to advance their simulation times with comparable speeds, thus are able to avoid wasting time and resources on overoptimistic executions and execution rollbacks. ArmVee is evaluated using a real-world simulation model with various simulation workload inputs and different parameter settings. The experimental results show that ArmVee is able to speed up the simulation execution significantly. In addition, it also greatly reduces memory usage and is scalable.

Categories and Subject Descriptors: 1.6.8 [**Simulation and Modeling**]: Types of Simulation—*Distributed*

General Terms: Design, Algorithms, Architecture, Mechanism, Performance

Additional Key Words and Phrases: Resource provisioning, virtual execution environments, parallel and distributed simulations, time synchronization, workload balance

ACM Reference Format:
Zengxiang Li, Wentong Cai, Stephen John Turner, Xiaorong Li, Ta Nguyen Binh Duong, and Rick Siow Mong Goh. 2015. Adaptive resource provisioning mechanism in VEEs for improving performance of HLA-based simulations. ACM Trans. Model. Comput. Simul. 26, 1, Article 1 (June 2015), 25 pages.
DOI: http://dx.doi.org/10.1145/2717309

1. INTRODUCTION

A parallel and distributed simulation [Fujimoto 2000], which is typically composed of a group of sequential simulations, is usually employed to study a complex system

This work is supported by the Future Data Center Technology Thematic Strategic Research Programme of the Singapore Agency for Science, Technology and Research (A*STAR) with grant number 112 172 0015.
Authors' addresses: Z. Li and R. S. M. Goh, Institute of High Performance Computing, Singapore, 138632; emails: {liz, gohsm}@ihpc.a-star.edu.sg; W. Cai and S. J. Turner, School of Computer Engineering, Nanyang Technological University, Singapore, 639798; emails: {aswtcai, assjturner}@ntu.edu.sg; X. Li, Infocomm Development Authority of Singapore, Singapore, 117438; email: Xiaorong_LI@ida.gov.sg; T. N. B. Duong, Ngee Ann Polytechnic, Singapore, 599489; email: tnb2@np.edu.sg.
Permission to make digital or hard copies of part or all 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 show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2015 ACM 1049-3301/2015/06-ART1 $15.00
DOI: http://dx.doi.org/10.1145/2717309

in desired fidelity or to enable collaboration among a number of geographically distributed participants. In order to support interoperability and reusability of simulation components, the *High-Level Architecture* (HLA), IEEE 1516 standard [IEEE 2010], is proposed. It provides a general framework to build a parallel and distributed simulation (*federation*) by integrating various simulation components (*federates*). Since the 1990s, parallel and distributed simulations (or HLA-based simulations) have been widely used in both civilian applications (e.g., transportation optimization [Suh et al. 2014]) and noncivilian applications (e.g., military training [Hannay et al. 2014]).

One of the greatest challenges in parallel and distributed simulations is time synchronization, which ensures that events, either generated by the federate itself (internal events) or received from other federates (external events), are processed in *timestamp* (TS) order. Time synchronization can be conducted in either conservative [Bryant 1977; Chandy and Misra 1979] or optimistic [Jefferson 1985] manners. In this article, we focus on optimistic synchronization, which allows a federate to process events and to advance simulation time freely. However, the faster federate (in terms of simulation time) may conduct overoptimistic executions and roll back its execution on receiving messages from the slower federate. It is generally agreed that optimistic synchronization has good performance when all federates have comparable execution speeds (i.e., how fast the simulation time is advanced) [D'Angelo 2011]. This usually requires that federates should have a balanced simulation workload. Unfortunately, it is very difficult to meet such a requirement in practice. Federates modeling different parts of the simulated system may have different simulation workloads. For instance, federates might have different number of *Simulated Model Entities* (SMEs). Moreover, the workload of each federate might changes dynamically during the simulation execution.

Recently, we have witnessed an increased interest in moving parallel and distributed simulations to Cloud and data centers for the purpose of obtaining large amounts of resources at low prices. Different from the traditional execution platform, the typical building block inside Cloud or data centers is the multicore-based computers. For instance, a computer may be installed with 4 Intel Xeon processors, each of which may have 10 CPU cores. Thanks to the increased resources, a large number of federates are able to run on the same computer. In addition, virtualization technologies [Barham et al. 2003] are used heavily in Cloud and data centers. In *Virtual Execution Environments*(VEEs), federates are encapsulated and executed on the resident *virtual machines* (VMs) with their own guest operating systems. Therefore, federates developed by different participants using different operating systems can be consolidated in the same computer without leaking confidential information. By default, the federates will share the resources evenly, as their resident VMs are scheduled by the hypervisor using a fair-share scheduler. Subsequently, the simulation performance may degrade significantly due to the simulation workload imbalance in federates [Yoginath and Perumalla 2013]. To solve this problem, an *Adaptive Resource provisioning Mechanism in Virtual Execution Environments* (ArmVee) is proposed in this article.

Our proposed ArmVee is composed of two modules: performance monitor and resource manager. The performance monitor, using a middleware approach, measures federate performance (i.e., execution speed) transparently to the simulation application. The resource manager is able to limit each VM to a certain resource share (e.g., percentages of scheduling time slots of one CPU core) by setting the parameters of the default fair-share scheduler in the hypervisor. Hence, it can control the execution speed of federates in a fine-grained manner. The resource manager periodically retrieves performance measurements of federates from performance monitors and fetches the available resources from hypervisor. A self-adaptive auto-regressive-moving-average (ARMA) model, commonly used in control theory, is adopted to capture the relationship between federate performance and the resource share of the resident VM. Based on the

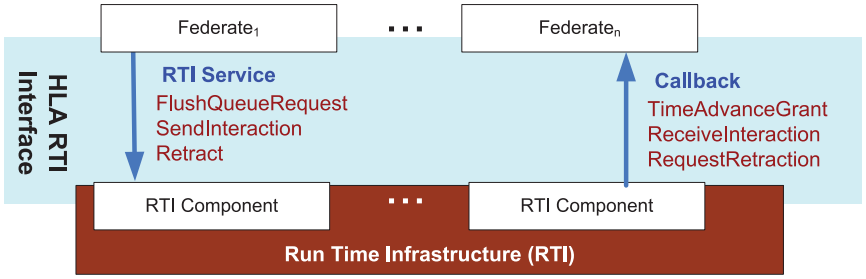


Fig. 1. Overview of an HLA-based simulation and the HLA RTI interface for optimistic synchronization.

ARMA model, the resource manager is able to distribute the available resources among the VMs, allowing their corresponding federates to have comparable execution speed. Generally, the higher the simulation workload, the greater resource share the federate will be allocated. Since federates are proactively controlled to advance simulation time with comparable speeds, they can avoid overoptimistic executions and execution rollbacks. Therefore, the execution time of the whole simulation can be reduced.

In summary, ArmVee solves the problem of workload imbalance by reallocating resources dynamically toward federates according to fluctuations in their simulation workloads. Furthermore, ArmVee avoids recoding federates and is transparent to the simulation application. It does not interrupt the simulation execution and does not introduce additional overhead on data transfer.

As an extension of our previous work [Li et al. 2013], this article adopts the ARMA model to replace the naive model used in the previous work, as the ARMA model captures more accurately the relationship between federate performance and VM resource share in the presence of dynamic simulation workloads. Consequently, a new algorithm is needed for the resource manager to distribute the available resources among federates. In addition, this article provides theoretical analysis concerned with the advantages of ArmVee on both speeding up simulation execution and reducing memory usage. Compared with our previous work [Li et al. 2013], more experimental results are reported to compare ARMA and naive models, to verify our theoretical analysis concerned with the advantages of ArmVee, and to investigate the scalability of ArmVee. Last, but not least, this article provides a more comprehensive literature review, for example, the related work on speeding up simulation on multicore-based computers.

The rest of the article is organized as follows: Section 2 introduces background knowledge on HLA-based simulations and resource provisioning in VEEs. Section 3 discusses related work on speedup parallel and distributed simulations. Section 4 illustrates the details of ArmVee, including the performance monitor and the resource manager. Section 5 analyzes the advantages of ArmVee theoretically. Section 6 describes the experiment design and discusses the experimental results. Finally, Section 7 concludes the paper and outlines some future work.

2. BACKGROUND

2.1. HLA-Based Simulations

As shown in Figure 1, an HLA-based simulation (federation) is composed of a group of simulation components (federates). The HLA standard defines rules, formats, and interfaces to support interoperability and reusability of the federates, which might be developed by different participants from different organizations. The *Runtime Infrastructure* (RTI), a communication middleware, implements the HLA interface specification. An RTI component is provided by the RTI to connect each joined federate

to the federation. It takes care of the federate's need to exchange messages and to request time advances. A federate invokes RTI services of its RTI component, while the RTI component delivers callbacks to the federate. For instance, a federate sends a message by invoking a *SendInteraction* RTI service, and the message is delivered to a receiving federate in the form of a *ReceiveInteraction* callback.

The HLA standard supports optimistic [Jefferson 1985] synchronization through the interfaces shown in Figure 1. An optimistic federate invokes *Flush Queue Request* (FQR) service iteratively for a time advancement request. It forces the RTI to deliver all buffered messages to the federate. In the mean time, the RTI may send a *TimeAdvanceGrant* to the federate after a federation-wide time synchronization. The granted time, referred to as *logical time* of the federate, defines the lower bound of future execution rollbacks. It can be used for fossil collection, that is, the memory space containing checkpointed states with TS smaller than logical 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. Hence, it might receive a straggler message whose TS is smaller than its simulation time. If so, a causality error occurs and the federate needs to roll back its execution by discarding the *overoptimistic execution* (i.e., the execution of those events with TS greater than the TS of the straggler message). It must undo the modification on the federate state using a federate state–saving and restoration mechanism [Jefferson 1985; Fujimoto 2000]. Additionally, it needs to invoke *Retract* services to unsend those incorrect messages that were sent during the overoptimistic execution. In the case that the messages have 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.

2.2. Resource Provisioning Mechanisms in VEEs

In Cloud and data centers, underprovisioning of resources may cause *Service Level Agreement* (SLA) violations, resulting in financial penalties; overprovisioning of resources will increase cost and waste resources [Wesam et al. 2012]. To avoid these problems, several adaptive resource provisioning mechanisms have been proposed by exploiting the advantages of virtualization technology. Applications are allowed to rent VMs according to their dynamic workload and Quality of Service (QoS) requirements [Duong et al. 2012]. They are able to choose VMs with different capacities/prices and bootup/shutdown VMs if necessary [Yang et al. 2012].

In contrast, ArmVee proposed in this article enables adaptive resource provisioning by adjusting the capability of individual VMs dynamically according to their overlying application workloads. In VEE, multiple VMs can be consolidated on the same computer with multicore CPU and large memory. They are scheduled by a hypervisor to share the underlying resources. Currently, we are focusing on the CPU resource that is most important to compute-intensive simulations. ArmVee is implemented based on the native CPU scheduler in Xen, that is, credit scheduler [Xen 2013]. By default, the credit scheduler is a fair-share scheduler, as all VMs are assigned the same credit. However, the assigned credit can be adjusted by setting the parameters (i.e., *weight* and *cap*) of the credit scheduler [Xen 2013]. In the case that only the *weight* parameter is set for each VM, the credit scheduler is work-conserving, which means that a VM that has spent all of its credit will be allocated additional CPU share if there is available CPU resource. Optionally, we can use the *cap* parameter to specify the maximum CPU share the VM is allowed to consume. In this case, the credit scheduler is nonwork-conserving, which means that a VM never consumes CPU share beyond the cap even if there is available CPU resource. Compared with the work-conserving scheduler, the nonwork-conserving scheduler provides better performance isolation among VMs

[Schanzenbach and Casanova 2008; Barker and Shenoy 2010]. The cap value, rather than the weight value, precisely specifies the CPU share (percentages of time slots) allocated to a VM. In addition, changing the cap value does not introduce additional overhead, and the new cap value can take effect immediately [Schanzenbach and Casanova 2008]. In this article, the CPU resources allocated to federates are controlled through the fine-grained adjustment of the caps of their resident VMs. Since the cap parameter can be adjusted through a hypercall (i.e., hypervisor API), ArmVee does not require any modification on the hypervisor.

3. RELATED WORK

A number of research works have been conducted improving performance of parallel and distributed simulations on traditional execution platforms. To achieve workload balance, federates [Yuan et al. 2004; Li et al. 2007] or their SMEs [Bononi et al. 2006] are migrated between computers. However, these migration protocols require recoding federates to provide state-saving and restoration functions and to enable communication redirection. In addition, the migration overhead is usually considerable due to the huge data transfer. Federates are interrupted in their executions. This may further block the execution of the entire federation.

The performance of parallel and distributed simulations can also be improved by efficient time synchronization. Some optimization protocols, e.g., *Moving Time Window* protocol [Sokol et al. 1988] and *Adaptive Time Warp* protocol [Panesar and Fujimoto 1997], have been proposed to improve performance of optimistic synchronization by blocking optimistic executions reactively. Specifically, a simulation component is blocked if it has executed far beyond others in the same simulation execution. However, the performance of these optimization protocols are sensitive to the frequency of global synchronization. Moreover, resources are wasted as some computers are set to idle during the simulation execution. Different from these optimization protocols, ArmVee proactively controls the federates to advance simulation time with comparable speeds, thus avoiding overheads on overoptimistic execution and execution rollbacks.

Recently, Cloud and data centers have become new emerging execution platforms for parallel and distributed simulations [Fujimoto et al. 2010]. A workload balance mechanism [D'Angelo 2011] and an optimized synchronization protocol [Malik et al. 2009] were proposed to speed up simulation execution in the novel execution platforms.

Multicore-based computers, the building blocks of Cloud and data centers, have increasing computation and communication capacities. In order to explore these capacities, researchers have proposed designs for individual federates in a multithreaded manner [Carothers et al. 2000]. Since all activities performed by the federate are executed by worker threads, it has the potential to enhance the internal parallelism and reduce the communication overhead [D'Angelo et al. 2012; Jagtap et al. 2012]. In addition, solutions to workload imbalance were proposed by global event scheduling [Chen et al. 2011] or by reallocating different numbers of CPU cores to federates according to their simulation workloads [Vitali et al. 2012]. In this article, federates, as sequential simulation components [Fujimoto 2000] are developed and executed as single-threaded processes [Martin et al. 1996]. How to extend ArmVee to multithreaded federates will be discussed in future work.

In addition, Child and Wilsey [2012] have proposed speeding up parallel and distributed simulations relying on *Dynamic Voltage and Frequency Scaling* (DVFS), which is commonly supported in multicore-based computers. They control federate performance by adjusting the frequency and voltage of its corresponding CPU core. However, due to the hardware limit, only four to six discrete frequency-voltage pairs are made possible. In contrast, the virtualization technologies used in this article enable fine-grained adjustment of CPU resources.

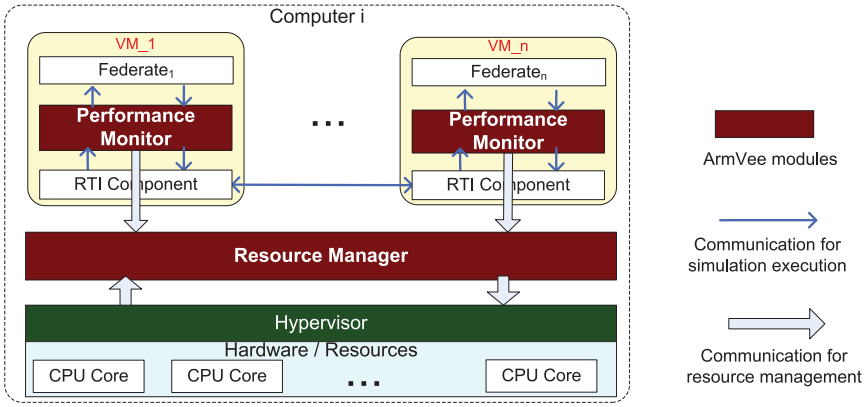


Fig. 2. Adaptive Resource provisioning Mechanism in Virtual Execution Environments (ArmVee).

To the best of our knowledge, limited work has been conducted to speed up parallel and distributed simulations by harnessing virtualization technologies that are widely deployed in Cloud and data centers. Yoginath and Perumalla [2013] have proposed a global VM scheduler that collects the simulation time of all federates and schedules their resident VMs in least-simulation-time-first order. Similar to ArmVee, the global VM scheduler can improve synchronization efficiency. However, its implementation is nontrivial, as the simulation application, the guest operating system, and the VM scheduler in hypervisor must be modified. In contrast, ArmVee is implemented in a transparent manner.

Similar to ArmVee, the adaptive resource-provisioning mechanisms proposed in Gong et al. [2010], Shen et al. [2011], Kalyvianaki et al. [2009], and Padala et al. [2009] adjust VM capabilities dynamically. However, they are targeted at server applications and cannot be applied directly for optimistic parallel and distributed simulations. For instance, the workload of server applications can be predicted according to historical CPU utilization rates [Gong et al. 2010; Shen et al. 2011], as the CPU is put in idle status when the workload is light. In contrast, optimistic federates consume allocated CPU resources on optimistic executions that might be rolled back in the future. Hence, the CPU utilization rate does not represent the useful simulation workload of the federate, which should not include the overoptimistic executions and execution rollbacks. In addition, the performance targets of server applications are well defined in SLA and their real performance can be measured easily [Kalyvianaki et al. 2009; Padala et al. 2009]. In contrast, the performance of a parallel and distributed simulation depends on the performance of individual federates, as well as the time synchronization among them. Moreover, it is nontrivial to measure federate performance transparently to the simulation application.

4. ADAPTIVE RESOURCE PROVISIONING MECHANISM IN VIRTUAL EXECUTION ENVIRONMENT (ARMVEE)

4.1. Mechanism Overview

The overview of ArmVee is illustrated in Figure 2. Federates are encapsulated and executed on their resident VMs. A number of VMs can be consolidated in the same multicore-based computer. The ArmVee is composed of two modules: performance monitor and resource manager.

The performance monitor, as middleware, is inserted between the federate and its RTI component. Similar to the solutions in Wang et al. [2005] and Quaglia [2006], the

performance monitor is transparent to both the simulation application and the RTI. By intercepting the RTI services and callbacks, the performance monitor is able to measure the execution speed of federates.

The resource manager is developed as a bridge between the performance monitors of federates and the hypervisor of the computer. It periodically adjusts the resource share of VMs according to the simulation workload of their corresponding federates. The simulation execution is divided into a number of successive control intervals and the control interval length is a parameter used in ArmVee. In each control interval, the resource manager retrieves execution speeds of federates through their performance monitors and the available resources of the computer through its hypervisor. A prediction model commonly used in control theory is adopted to capture the relationship between the performance of a federate and the resource share (i.e., cap value) of its resident VM. Consequently, the resource manager is able to calculate appropriate cap values of those VMs, to make federates with different simulation workloads achieve comparable execution speeds.

Compared with the scheme that evenly distributes CPU sources among federates, ArmVee allocates less CPU resources to federates with lower workload. Hence, they can avoid overoptimistic executions and execution rollbacks. On the other hand, ArmVee allocates more CPU resources to federates with higher workload. Hence, the execution time of the whole federation can be reduced. In theory, ArmVee is able to ensure that federates advance simulation time with the same speed. Hence, federates never waste CPU resources on the overoptimistic execution and execution rollbacks. From this point of view, the federation would be able to obtain the optimum execution speed with the given CPU resources.

In this article, we assume that all federates in the federation are consolidated on one computer. In the future, we will extend our work for large-scale HLA-based simulations executed on multiple computers. The computers are coordinated to ensure that all federates have comparable execution speeds. In addition, some VMs may be migrated among computers [Clark et al. 2005] if necessary to achieve workload balance.

4.2. Performance Monitor

Intuitively, the execution speed of a federate can be measured as the advanced simulation time in a control interval divided by the control interval length. However, execution rollbacks might occur during a control interval. If so, the federate decreases its simulation time and wastes processing time on overoptimistic executions and execution rollbacks. Hence, the execution speed measured in this manner may not accurately characterize the useful simulation 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 are usually much smaller than the control interval. Only the epochs without overoptimistic executions and execution rollbacks are taken into account for performance measurement.

As shown in Figure 2, the performance monitor as the middleware is able to intercept the FQR services that are invoked by federates iteratively. Hence, an epoch can be bounded by two subsequent FQR services. The requested simulation times of these FQR services define the *begTS* and *endTS* of the epoch, respectively. Consequently, the performance monitor can calculate the advanced simulation time (*advTS*) and the elapsed execution time (*exeTime*) in the epoch. The epochs in current control interval are kept in a list denoted as **E**. In addition, the performance monitor can also intercept the messages and RR callbacks delivered to the federate. In the case that there is a message or RR callback with *TS* smaller than the *endTS* of an epoch in **E**, the epoch must include overoptimistic execution. It is removed from **E**, as it should not be taken

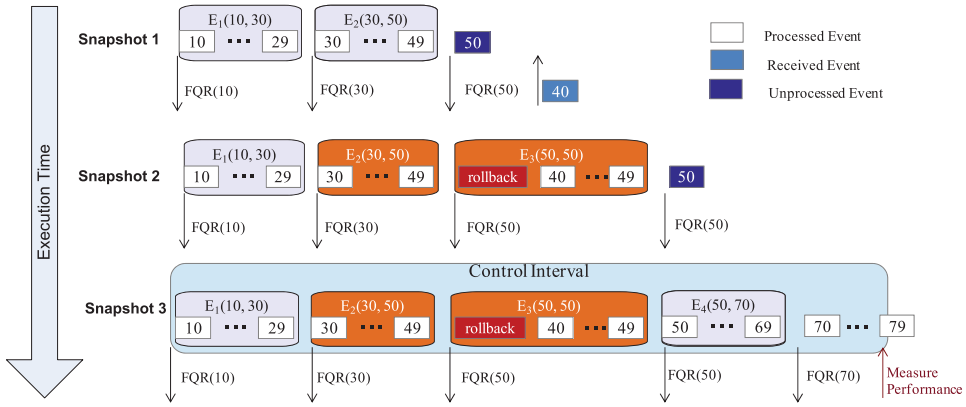


Fig. 3. Example of performance measurement of a control interval in performance monitor.

into account for performance measurement. In the meantime, the epochs in which execution rollback occurs are also removed from **E**.

At the end of each control interval, the performance monitor is required to measure federate execution speed during the control interval (Algorithm 1). 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 value. 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.

ALGORITHM 1: Performance Monitor: Measure Execution Speed

1: **if E** is empty **then**
2: $exeSpeed = InvalidValue$
3: **else**
4: $exeSpeed = \frac{\sum_{E_i \in \mathbf{E}} E_i \rightarrow advTS}{\sum_{E_i \in \mathbf{E}} E_i \rightarrow exeTime}$
5: **end if**
6: Remove all elements from **E**
7: **return** *exeSpeed*

Figure 3 illustrates three snapshots of an optimistic federate during its execution. As shown in Figure 2, the performance monitor intercepts the FQR services invoked by the federates and the messages delivered to the federate. At the end of each control interval, the resource manager informs the performance monitor to measure federate performance and thus adjust the resources of all VMs periodically. The numbers in the figure might be the TS of events, requested time of FQR invocations, *begTS* or *endTS* of epochs. Usually, federates are not required to invoke an FQR service before processing every event. That is, an epoch created in the performance monitor may include multiple events. Epochs E_1 and E_2 are created on intercepting $FQR(30)$ and $FQR(50)$, respectively. Before delivering the event with TS equal to 40 to the federate, an execution rollback is detected. E_2 is not considered for performance measurement, as the execution from 40 to 49 is an overoptimistic execution. In the subsequent epoch E_3, the federate handles the execution rollback before processing events from 40 to 49. Hence, E_3 is not considered for performance measurement either. After that, E_4 is

created on intercepting $FQR(70)$. On receiving the performance measurement request, only E_1 and E_4 are considered for execution speed calculation in Algorithm 1.

4.3. Resource Manager

4.3.1. ARMA Model. In each control interval, the resource manager retrieves the performances of federates from their performance monitors and the available resources of the computer from its hypervisor. It is responsible for distributing the available resources to federates and standardizes their performance. Due to the dynamic simulation workload, the relationship between federate performance and VM cap is often nonlinear. Since a nonlinear model can lead to unacceptable complexity and runtime overhead, a common method used in control theory is to approximate the relationship locally by a linear model. Similar to Kalyvianaki et al. [2009] and Padala et al. [2009], a self-adaptive ARMA model is adopted to capture the relationship between federate performance and VM cap in the presence of dynamic simulation workload.

Take the i^{th} federate (F_i) as an example; at the k^{th} control interval, we can predict the execution speed of F_i (denoted as $PES_i(k)$) using the ARMA model:

$$PES_i(k) = \sum_{j=1}^{m} \alpha_{ij} \times MES_i(k-j) + \sum_{j=0}^{n} \beta_{ij} \times C_i(k-j), \tag{1}$$

where $MES_i(k-j)$ and $C_i(k-j)$ denotes, respectively, the measured execution speed and VM cap of F_i at the $(k-j)^{th}$ control interval. Note that the auto-regressive weight (α_{ij}) and the moving-average weight (β_{ij}) are updated in every control interval based on the data collected in the past intervals. In our experiments (Section 6), the auto-regressive term (m) is set to 2 and the moving-average term (n) is set to 1. These settings achieve reasonable trade-off between model accuracy and model complexity. Since $MES_i(k-1)$, $MES_i(k-2)$, and $C_i(k-1)$ are known, Equation (1) is simplified as:

$$PES_i(k) = \beta_{i0} \times C_i(k) + \Delta_i(k), \tag{2}$$

where $\Delta_i(k) = \alpha_{i1} \times MES_i(k-1) + \alpha_{i2} \times MES_i(k-2) + \beta_{i1} \times C_i(k-1)$.

To speed up the simulation execution, the PES of all federates should be maximized in every control interval while satisfying the constraints outlined later. First, the caps of VMs should be constrained in a predefined range. That is,

$$C_{low} \le C_i(k) \le C_{high} \quad \left(1 \le i \le N, 1 \le k \le \frac{T}{t}\right), \tag{3}$$

where N denotes the federation scale, T denotes the execution time of the whole simulation, and t denotes the control interval length. Generally, $C_{low} \ll C_{high}$. In our experiments, C_{low} is equal to 5. It ensures that federates can respond in time for the federation-wide synchronization. C_{high} is equal to 100, as federates (single-threaded processes [Martin et al. 1996]) cannot consume more than one CPU core.

Second, the federates can consume only the available CPU resources on the computer. For instance, if there are M CPU cores available, the total cap of VMs should not be greater than $CAP = M \times 100$. That is,

$$\sum_{i=1}^{N} C_i(k) \le CAP \quad \left(1 \le k \le \frac{T}{t}\right). \tag{4}$$

Generally, $N \times C_{low} \ll CAP \ll N \times C_{high}$.

Third, all federates in the federation should have the same *PES* in the same control interval. That is,

$$PES_i(k) = PES(k) \quad \left(1 \le i \le N, 1 \le k \le \frac{T}{t}\right). \tag{5}$$

Due to the prediction error in the ARMA model and the VM scheduling overhead in the hypervisor, the real execution speed of federates might be slightly different from the predicted execution speed, that is, $MES_i(k) \approx PES_i(k)$. Therefore, federates in the federation have comparable execution speeds, thus the synchronization overhead can be reduced significantly.

ALGORITHM 2: Resource Manager: The k^{th} Control Interval

1: $\mathbf{F} = \{F_1, F_2, \ldots, F_N\}$;
2: **for** each $F_i \in \mathbf{F}$ **do**
3: $MES_i(k-1) = F_i \rightarrow measurePerformance()$;
4: **if** $MES_i(k-1)$ *is an invalid value* **then**
5: $MES_i(k-1) = PES_i(k-1)$;
6: **end if**
7: **end for**
8: **for** each $F_i \in \mathbf{F}$ **do**
9: $PES_i(k, C_{low}) = \beta_{i0} \times C_{low} + \Delta_i(k)$;
10: $PES_i(k, C_{high}) = \beta_{i0} \times C_{high} + \Delta_i(k)$;
11: **end for**
12: Choose $F_v \in \mathbf{F}$ where $PES_v(k, C_{low}) = \max_{F_i \in \mathbf{F}} PES_i(k, C_{low})$;
13: Choose $F_u \in \mathbf{F}$ where $PES_u(k, C_{high}) = \min_{F_i \in \mathbf{F}} PES_i(k, C_{high})$;
14: $PES_{low}(k) = PES_v(k, C_{low})$;
15: $PES_{high}(k) = PES_u(k, C_{high})$;
16: Call extreme case handler (i.e., Algorithm 3)
17: **while** $PES_{high}(k) - PES_{low}(k) > \varepsilon$ **do**
18: $PES(k) = \frac{PES_{high}(k) + PES_{low}(k)}{2}$;
19: **for** each $F_i \in \mathbf{F}$ **do**
20: $C_i(k) = \frac{PES(k) - \Delta_i(k)}{\beta_{i0}}$;
21: $PES_i(k) = PES(k)$;
22: **end for**
23: **if** $\sum_{F_i \in \mathbf{F}} C_i(k) < CAP$ **then**
24: $PES_{low}(k) = PES(k)$;
25: **else**
26: $PES_{high}(k) = PES(k)$;
27: **end if**
28: **end while**

The functionality of the resource manager is illustrated in Algorithm 2. $\mathbf{F}$ denotes the set of all federates in the federation. At the beginning of each (k^{th}) control interval, the resource manager retrieves *MES* of all federates at the previous ($(k-1)^{th}$) interval from their performance monitors (Lines 2 to 7). In the case that the retrieved *MES* of a federate (F_i) is an invalid value, the *MES* of the federate is set approximately as the *PES* of the federate in the previous control interval, that is, $MES_i(k-1) = PES_i(k-1)$.

Based on these performance measurements, the resource manager distributes available resources to federates in the federation for the purpose of maximizing *PES* of all federates while satisfying aforementioned constraints. In other words, the resource manager searches for the optimum setting of $C_i(k)$, which maximizes $PES(k)$, and in the meantime satisfies Equations (3), (4), and (5).

Intuitively, the higher the cap value of the VM, the higher the performance of the corresponding federate. Hence, $PES_i(k)$ increases with the increasing $C_i(k)$, and

$\beta_{i0} > 0$. To meet the first constraint (Equation (3)), $PES_i(k)$ must be in the range of $[PES_i(k, C_{low}), PES_i(k, C_{high})]$, where $PES_i(k, C_{low})$ (or $PES_i(k, C_{high})$) denotes the PES of the i^{th} federate during the k^{th} control interval if the VM cap is set to C_{low} (or C_{high}). That is, $PES_i(k, C_{low}) = \beta_{i0} \times C_{low} + \Delta_i(k)$ and $PES_i(k, C_{high}) = \beta_{i0} \times C_{high} + \Delta_i(k)$. In order to meet the third constraint (Equation (5)), we see that $PES(k) \geq PES_i(k, C_{low})$ and $PES(k) \leq PES_i(k, C_{high})$ for each $F_i \in \mathbf{F}$. Hence, $PES(k) \geq \max_{F_i \in \mathbf{F}} PES_i(k, C_{low})$ and $PES(k) \leq \min_{F_i \in \mathbf{F}} PES_i(k, C_{high})$. Let's denote the federates that have maximum $PES_i(k, C_{low})$ and minimum $PES_i(k, C_{high})$ as F_v and F_u, respectively (Lines 12 and 13). That is, F_v (F_u) has the lowest (highest) workload in the federation. According to Equation (3), we can get $C_v \geq C_{low}$ and $C_u \leq C_{high}$, thus, $PES_v(k) \geq PES_v(k, C_{low})$ and $PES_u(k) \leq PES_u(k, C_{high})$. To satisfy Equation (5), that is, $PES(k) = PES_v(k) = PES_u(k)$, we can deduct that $PES_v(k, C_{low})$ and $PES_u(k, C_{high})$ are the lower bound and upper bound of $PES(k)$, respectively, denoted as $PES_{high}(k)$ and $PES_{low}(k)$ (Lines 14 and 15). After that, a bisection method (Lines 17 to 28) is used to numerically search for the maximum value of $PES(k)$, which meets the second constraint (Equation (4)).

Before using the bisection method, the resource manager should handle the following two extreme cases using Algorithm 3.

a. *$PES_{low}(k)$ is greater than $PES_{high}(k)$*, that is, F_v (the federate with lowest workload) is still faster than F_u (the federate with highest workload) even if $C_v = C_{low}$ and $C_u = C_{high}$. This case seldom happens as $C_{low} \ll C_{high}$. If this case occurs, the third constraint (Equation (5)) cannot be satisfied. We simply set C_v to C_{low}, then distribute the remaining CPU resources among other federates (Lines 3 and 4). By iteratively removing F_v from $\mathbf{F}$, $PES_{low}(k)$ (i.e., $\max_{F_i \in \mathbf{F}} PES_i(k, C_{low})$) decreases; finally, we get $PES_{low}(k) \leq PES_{high}(k)$ and terminate the iterative procedure.

b. *$\sum_{F_i \in \mathbf{F}} C_i(k) < CAP$ when $PES(k) = PES_{high}(k)$*, that is, federates cannot consume all available CPU resources, although their execution speed is equal to the maximum value $PES_u(k, C_{high})$. This case seldom occurs, as $CAP \ll N \times C_{high}$. If this case occurs, we simply set C_u to C_{high}, then distribute the remaining CPU resources among other federates (Lines 12 and 13). By iteratively removing F_u from $\mathbf{F}$, $PES_{high}(k)$ increases. The iterative procedure will be terminated if $\sum_{F_i \in \mathbf{F}} C_i(k) \geq CAP$ or $\mathbf{F} = \emptyset$. Similar to *Case a*, the third constraint (Equation (5)) cannot be satisfied either.

ALGORITHM 3: Extreme Case Handler

```
while PES_low(k) > PES_high(k) do
  C_v(k) = C_low;
  F = F − F_v;
  CAP = CAP − C_low;
  Line 12 and Line 14 from Algorithm 2
end while
while F ≠ ∅ do
  PES(k) = PES_high(k);
  Lines 19 to Line 22 from Algorithm 2
  if ∑_{F_i∈F} C_i(k) < CAP then
    C_u(k) = C_high;
    F = F − F_u;
    CAP = CAP − C_high;
    Line 13 and Line 15 from Algorithm 2
  else
    Break;
  end if
end while
```

4.3.2. Naive Model. Previously, we have deployed a naive model to capture the relationship between federate execution speed and VM cap value [Li et al. 2013]. In the naive model, we assume that the simulation workload in the k^{th} control interval is the same as that in the $(k-1)^{th}$ control interval and that federate execution speed increases proportionally with the increasing VM cap value, that is,

$$PES_i(k) = MES_i(k-1) \times \frac{Cap_i(k)}{Cap_i(k-1)}. \tag{6}$$

Hence, the CPU resources can be distributed proportionally among federates to get the maximum $PES(k)$, while satisfying aforementioned constraints (i.e., Equations (3), (4) and (5)). Compared with the naive model, the ARMA model adopted in this article uses more historical data, taking the dynamic simulation workload into account. For this reason, the ARMA model is expected to be more accurate than the naive model to capture the relationship between federate execution speed and VM cap value. Hence, it is expected to further speed up the simulation execution (refer to experimental results reported in Section 6.2.1).

5. ADVANTAGES OF ARMVEE

5.1. Accelerating Simulation Execution

Without ArmVee, the available CPU resources are evenly distributed among federates, that is, $C_i = \overline{C} = \frac{CAP}{N}$. As federates have different simulation workload, their execution speed measured by performance monitors are different. Suppose that F_l is the slowest federate, that is, $MES_l = min_{F_i \in \mathbf{F}} MES_i$. Then, MES_i can be denoted as $\omega_i \times MES_l$, where $\omega_i \geq 1$. Since the federation execution is determined by the slowest federate, the federation execution speed is equal to MES_l.

With ArmVee, the CPU resources are distributed among federates according to their simulation workload. It guarantees that federates have the same execution speed, ignoring the prediction error of the ARMA model, the extreme cases handled in Algorithm 3, and the VM scheduling overhead in hypervisor. Suppose that the execution speed of F_i and the cap value of its resident VM are denoted as MES'_i and C'_i, respectively. Then we get, $MES'_i = MES'_l$ and $\sum_{F_i \in \mathbf{F}} C'_i = CAP$. According to the performance evaluation in Schanzenbach and Casanova [2008], the compute rate of VM increases proportionally with increasing cap value. Hence, for the same simulation workload on F_l and F_i, we get

$$\frac{MES'_l}{MES_l} = \frac{C'_l}{\overline{C}} \quad and \quad \frac{MES'_i}{MES_i} = \frac{C'_i}{\overline{C}}. \tag{7}$$

Therefore, we can deduce that $C'_i = \frac{C'_l}{\omega_i}$, thus,

$$\sum_{F_i \in \mathbf{F}} C'_i = C'_l \times \sum_{F_i \in \mathbf{F}} \frac{1}{\omega_i} = CAP. \tag{8}$$

Consequently, the federation execution speed is equal to MES'_l, and

$$MES'_l = \frac{MES_l \times C'_l}{\overline{C}} = \frac{MES_l \times CAP}{\overline{C} \times \sum_{F_i \in \mathbf{F}} \frac{1}{\omega_i}} = \frac{MES_l \times N}{\sum_{F_i \in \mathbf{F}} \frac{1}{\omega_i}}. \tag{9}$$

Therefore, the execution speedup of ArmVee (i.e., the ratio of federation execution speed with ArmVee vs. that without ArmVee) is

$$\frac{MES'_i}{MES_l} = \frac{N}{\sum_{F_i \in \mathbf{F}} \frac{1}{\omega_i}} \geq 1. \tag{10}$$

That is, ArmVee can always speed up the simulation execution. The more significant the workload imbalance among federates, the greater ω_i, thus the greater the execution speedup of ArmVee.

5.2. Reducing Memory Usage

In case of future execution rollbacks, an optimistic federate should save its execution state before processing events with TS greater than its logical time. Due to time synchronization, the logical time advancement is determined by the slowest federates in the federation. Without ArmVee, federates advance simulation time with different speeds. Hence, some federates may have simulation time much greater than their logical time. They keep a large number of snapshots of federate state in memory. With ArmVee, federates advance their simulation times with comparable speeds. Hence, their simulation times are close to their logical times. They keep a small number of snapshots of federate state in memory.

In optimistic synchronization, the infrequent state-saving approach [Lin et al. 1993; Fujimoto 2000] is commonly adopted. According to the analysis in Lin et al. [1993] and Fujimoto [2000], the higher the risk of execution rollbacks, the higher the optimum state-saving frequency. Since ArmVee is able to reduce the risk of execution rollbacks dramatically, the optimum state-saving frequency and the memory usage can be reduced accordingly. However, determining the optimum state-saving frequency is nontrivial and is out of the scope of this article.

6. EXPERIMENTS AND RESULTS

6.1. Experiment Design

A *Massively Multiplayer Online Games* (MMOGs) ecosystem simulation model is used in our experiments to evaluate our proposed ArmVee. Nae et al. [2008] have introduced the concepts of the MMOGs ecosystem. In the MMOGs ecosystem, game operators rent resources (CPU, Network, and Memory) from data centers for running the MMOGs servers. They are able to dynamically adjust the number of resources rented 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-renting schemes using different workload prediction algorithms and different resource-hosting policies. The resource-renting schemes can be evaluated using performance metrics such as resource overallocation and resource underallocation [Nae et al. 2008].

In the HLA-based simulation, each federate simulates all game servers in a data center. It generates a local event to simulate the performance of the game servers in each timestep (i.e., 10s). Each local event calculates the responding 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 timestep. The number of CPU resources required for processing the local event thus increases proportionally with the increasing number of players connected to the data center. In our experiments, the number of players connected to the data centers is retrieved from the trace of RuneScape [Nae et al. 2008]. They have a strong diurnal pattern. Therefore, the simulation workload of the corresponding federates also have a strong diurnal pattern. In the case in which the data centers are located in different time-zone regions, the corresponding federates may have different simulation workloads at the same simulation time.

In addition, federates might send external events to each other to simulate the communication between game servers located at different data centers. For instance, game servers may replicate some parts of their game state in different data centers

to tolerate unpredictable failures [Mason 2009]. Depending on different strategies, state replications might be triggered due to different situations and the replication frequency might vary in a large range. For simplicity, we assume that 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 is equal to the TS of the processing internal event plus a *lookahead* (*LA*) [Fujimoto 2000], indicating that the external event will affect the receiving federate after *LA* simulation time. $P_{ExternEvent}$ and *LA* are used as the parameters in the HLA-based simulation. In our experiments, we will evaluate the performance of a number of simulation executions with different parameter inputs, that is, different values of $P_{ExternEvent}$ and *LA*.

Optimistic synchronization is employed in the HLA-based simulation. Federates are allowed to process three events before invoking the next FQR service. They will trigger a federation-wide time synchronization if $simulation\ time > logical\ time + 1{,}000$. In addition, the infrequent state-saving approach described in Fujimoto [2000] and Lin et al. [1993] is adopted. Federates save their execution state after processing 15 events.

The simulation length is 3 days, including the first 2 days as the warm-up period. The simulation is executed on an RTI implemented by reusing the code in SOHR [Pan et al. 2007]. For efficiency consideration, RTI components are implemented as the libraries of their corresponding federates; the communication among RTI components is facilitated by JAVA socket instead of the heavy Grid service invocation. Experiments are carried out on a computer with 12 Intel Xeon 2.67GHz CPU cores, 24GB RAM, CentOS 6.2, and Xen 4.1.2. Each federate executes on a VM with one VCPU core, 2GB RAM and CentOS 6.2.

Two methods are investigated in our experiments. The *Fixed* method uses the default credit scheduler, which shares CPU resources among VMs evenly, that is, $C_i = \frac{CAP}{N}$. The *Adaptive* method employs ArmVee to adjust the caps of VMs according to simulation workload. Without explicit specification, ArmVee adopts the ARMA model proposed in this article rather than the naive model proposed in Li et al. [2013].

6.2. Experiment Results

Experiments are carried out to illustrate the advantages of ArmVee on both improving simulation performance (Section 6.2.1) and reducing memory usage (Section 6.2.2). The *Fixed* and *Adaptive* methods are compared in different simulation executions with various real simulation workloads and different parameter settings in the HLA-based simulation. After that, the scalability of ArmVee is investigated (Section 6.2.3). Finally, we verify the theoretical analysis in Section 5 using synthetic simulation workloads (Section 6.2.4).

6.2.1. Improving Simulation Performance. For simplicity, the federation scale is two (i.e., $N = 2$) and one CPU core is available for the simulation execution (i.e., $CAP = 100$). Three cases of simulation workload are introduced in the two federates. Figure 4 illustrates the numbers of players connected to their corresponding data centers, which are obtained from the trace of RuneScape [Nae et al. 2008]. In Case (i), the data centers are located in the regions of the same time zone. In Cases (ii) and (iii), the time difference between the data centers is around 6 and 12 hours, respectively.

In the first series of experiments, we compare the simulation execution performance in the *Fixed* and *Adaptive* methods with the workload cases mentioned earlier. In the Fixed method, VM caps are fixed at 50. In the *Adaptive* method, the VM caps (Figure 5) are adjusted according to the simulation workloads of their corresponding federates (Figure 4). In workload Case (i), federates have similar workload, thus each federate obtains around 50% of the CPU resource. In workload Cases (ii) and (iii), the federate with higher simulation workload obtains more CPU resource. The

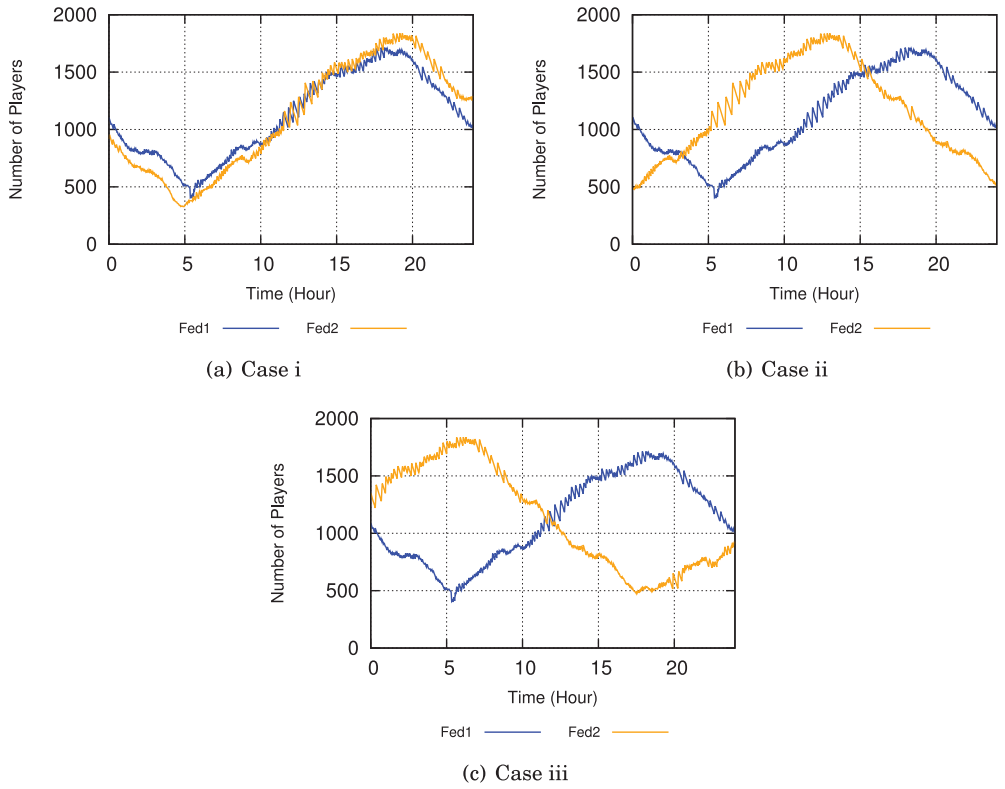
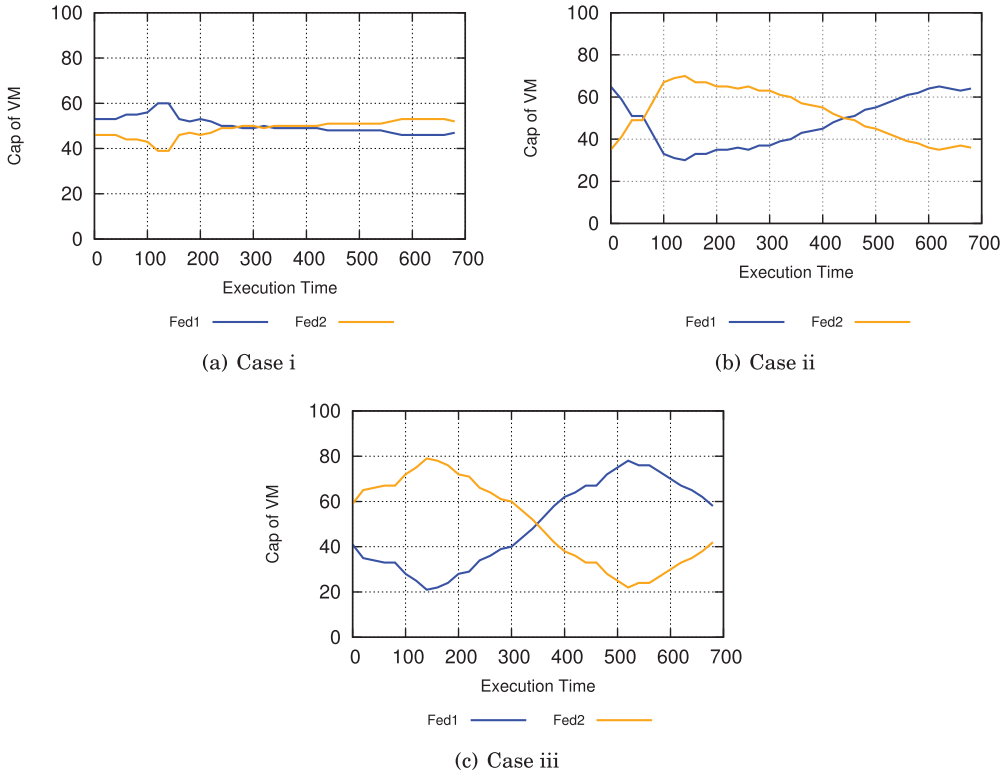


Fig. 4. Simulation workload of federates: number of players connected to their corresponding data centers.

greater the difference of simulation workload, the greater the difference of assigned VM caps.

The simulation execution performances are illustrated in Figure 6, including the simulation execution speed, the number of execution rollbacks, and the execution efficiency. The simulation execution speed is defined as the ratio of advanced simulation time (i.e., 1 day or 86,400s) to the simulation execution time. The execution efficiency is defined as the ratio of useful events to total events processed [Malik et al. 2009]. Besides the useful events, a federate might process events during over-optimistic executions and execution rollbacks. The *Fixed* method (*Adaptive* method) for workload Cases (i), (ii) and (iii) are denoted as $F(i)$, $F(ii)$, and $F(iii)$ ($A(i)$, $A(ii)$, and $A(iii)$), respectively, in the figure. The control interval length as the parameter of ArmVee ranges from 80 to 5s. This is meaningful for the *Adaptive* method only. In this series of experiments, the parameters of the HLA-based simulation are set as follows: $P_{ExternEvent} = 1\%$ and $LA = 45$. Different parameter settings will be studied later.

In workload Case (i), federates have similar workloads during the simulation execution. Consequently, federates are able to advance simulation time with comparable speeds. Hence, federates encounter a small number of execution rollbacks (Figure 6(b)); and they have high execution efficiency (Figure 6(c)). Therefore, we can observe from Figure 6(a) that the *Fixed* and *Adaptive* methods have similar execution speed. In workload Cases (ii) and (iii), federates have different workloads. For the *Fixed* method, compared with workload Case (i), the numbers of execution rollbacks are much greater (Figure 6(b)). Furthermore, the overoptimistic execution discarded in each execution



(a) Case i

(b) Case ii

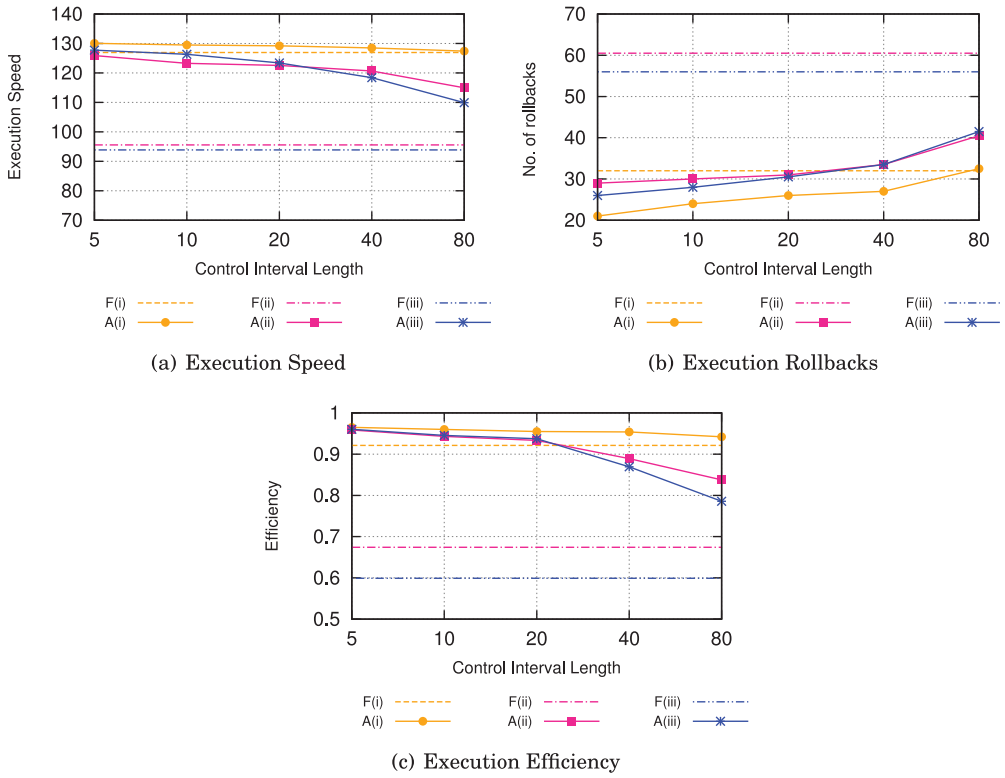
(c) Case iii

Fig. 5. VM Caps in Adaptive method when the control interval length is 20s.

rollback can be significant due to the different execution speeds of federates. As a result, the execution efficiency is much lower (Figure 6(c)). Therefore, we can observe from Figure 6(a) that the execution speeds of $F(ii)$ and $F(iii)$ are much lower than that of $F(i)$. Different from the *Fixed* method, federates in the *Adaptive* method have comparable execution speed. They encounter less execution rollbacks (Figure 6(b)) and have higher execution efficiency (Figure 6(c)). Therefore, ArmVee is able to speed up simulation execution significantly, as shown in Figure 6(a).

Generally speaking, the smaller the control interval length, the more accurately the resource manager adjusts resource shares according to simulation workload. As a result, federates encounter less execution rollbacks (Figure 6(b)) and have higher execution efficiency (Figure 6(c)). Therefore, we can observe from Figure 6(a) that the execution speed in the *Adaptive* method decreases with the increasing control interval length. When the control interval length is equal to 5s, the execution speedup of ArmVee (i.e., execution speed in the *Adaptive* method divided by that in the *Fixed* method) are 1.36 and 1.32 in workload Cases (ii) and (iii), respectively. It is worth pointing out that federates in the *Adaptive* method cannot avoid all execution rollbacks. This is because they cannot advance simulation time with exactly the same speed in practice, due to the prediction error in the ARMA model and the extreme cases handled in Algorithm 3.

The second series of experiments are carried out to compare the simulation execution performance in *Fixed* and *Adaptive* methods with different parameter settings in the HLA-based simulation. Due to the space limit, we only report the experimental results for the Case (iii) simulation workload, which is quite common in the MMOGs ecosystem, as the data centers are usually located at different places around the world. In



(a) Execution Speed

(b) Execution Rollbacks

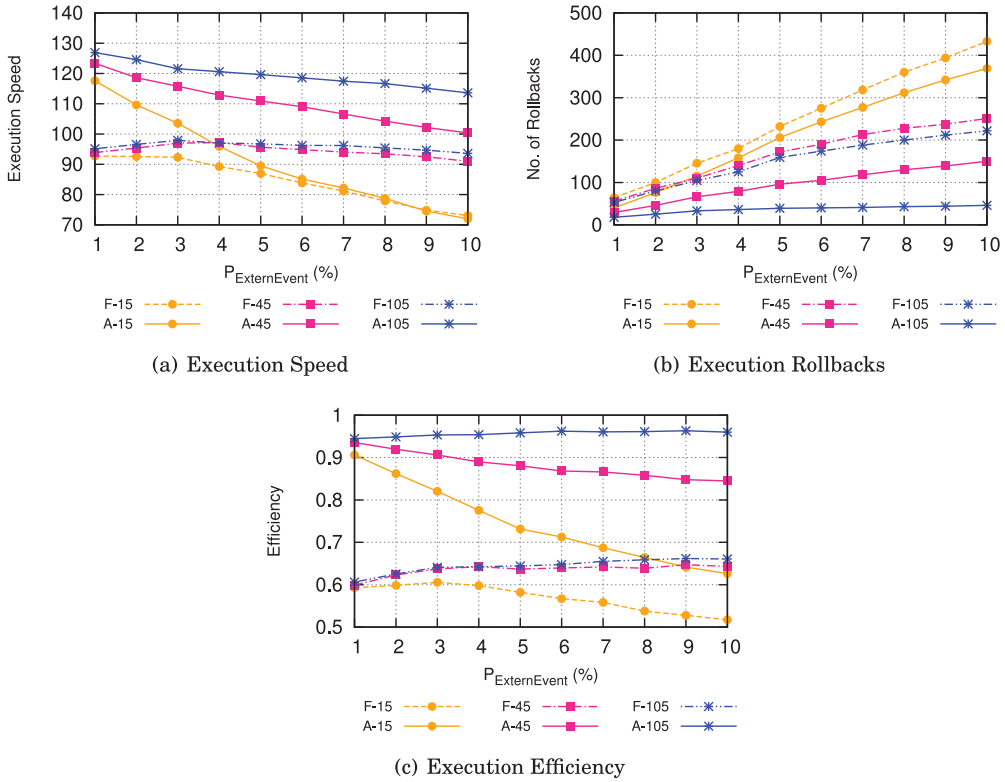
(c) Execution Efficiency

Fig. 6. Performance comparison between Fixed and Adaptive methods with various simulation workloads.

addition, the control interval length is set to 20s. Figure 7 illustrates the simulation execution speed, 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* methods for $LA = l$ are denoted as *A-l* and *F-l*, respectively.

In practice, ArmVee cannot ensure that federates advance their simulation time with the same speed. Federates are at risk for execution rollback if the difference of their simulation times is greater than *LA*. For this reason, when *LA* is too small (e.g., $LA = 15$), federates in the *Fixed* and *Adaptive* methods encounter a similar number of execution rollbacks (Figure 7(b)). In the case that $P_{ExternEvent}$ is small, the overoptimistic execution discarded in each execution rollback is much longer in the *Fixed* method than that in the *Adaptive* method. For this reason, the *Adaptive* method has a much higher execution efficiency (Figure 7(c)) and outperforms the *Fixed* method significantly (Figure 7(a)). When $P_{ExternEvent}$ increases, the overoptimistic execution in the *Fixed* method decreases as the faster federate is frequently rolled back by the slower federate due to the straggler messages. As a result, the difference in the execution efficiency and execution speed between *Adaptive* and *Fixed* methods decreases. The *Adaptive* method may even perform worse than the *Fixed* method when $LA = 15$ and $P_{ExternEvent} \geq 9\%$ (Figure 7(a)),[1] although the execution efficiency is still higher (Figure 7(c)). This is

[1]Xen 4.2 or later version provides a hypercall to change the timeslice of the CPU scheduler. Generally, the smaller the timeslice, the more frequently VMs are scheduled, and thus, the smaller difference in the simulation time of federates using the *Adaptive* method. According to our experiments, by reducing the



(a) Execution Speed

(b) Execution Rollbacks

(c) Execution Efficiency

Fig. 7. Performance comparison between Fixed and Adaptive methods with different parameter settings.

because, in the *Fixed* method, federates with low workload encounter overoptimistic executions and thus process much more events with small event processing time.

When LA is large (e.g., $LA = 105$), federates in the *Adaptive* method with comparable execution speed are able to avoid almost all rollbacks (Figure 7(b)). As a result, the execution efficiency in the *Adaptive* method is close to a value of one and is much higher than that in the *Fixed* method. 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 7(a) that ArmVee dramatically speeds up the simulation execution. When $LA = 45$, the *Adaptive* method can still outperform the *Fixed* method. However, the performance advantage decreases with the increasing $P_{ExternEvent}$, as the number of execution rollbacks increases and the execution efficiency decreases in the *Adaptive* method.

Figure 8 illustrates the execution speedup of ArmVee with respect to various parameter settings. As we can see, ArmVee achieves significant performance enhancement except for those parameter inputs $LA = 15$ and $P_{ExternEvent} > 6\%$. With the decreasing $P_{ExternEvent}$, the execution speedup increases as the overoptimistic execution in the *Fixed* method increases. With the increasing LA, the execution speedup increases as the number of execution rollbacks in the *Adaptive* method decreases. In the best cases, ArmVee is able to achieve 34% execution speedup.

timeslice from 30ms (the default value) to 10ms, the *Adaptive* method can outperform the *Fixed* method even if $LA = 15$ and $P_{ExternEvent} \geq 9\%$.

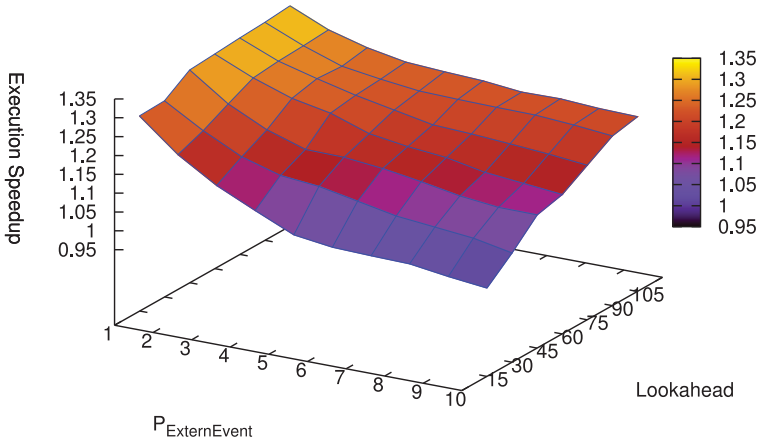
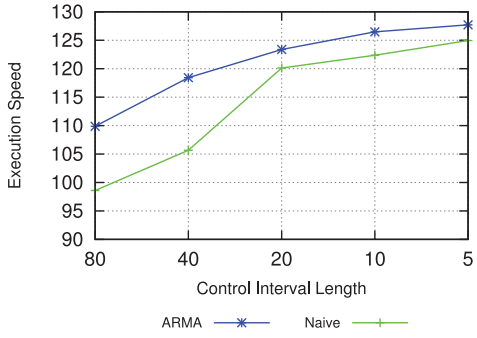
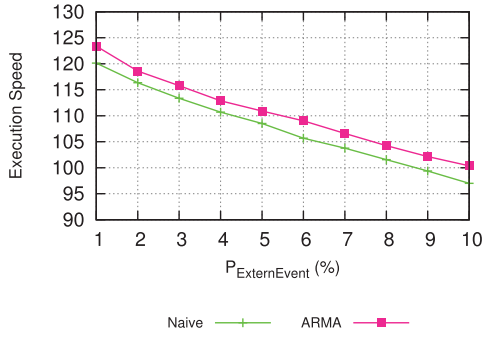


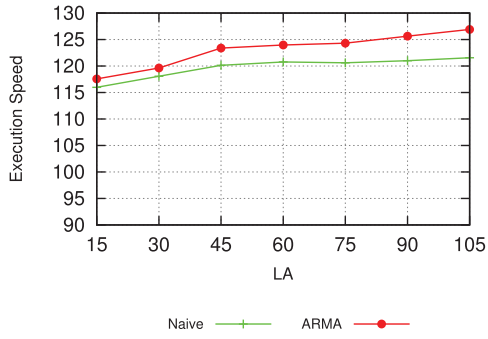
Fig. 8. Execution speedup of ArmVee.



(a) Execution speed v.s. control interval length



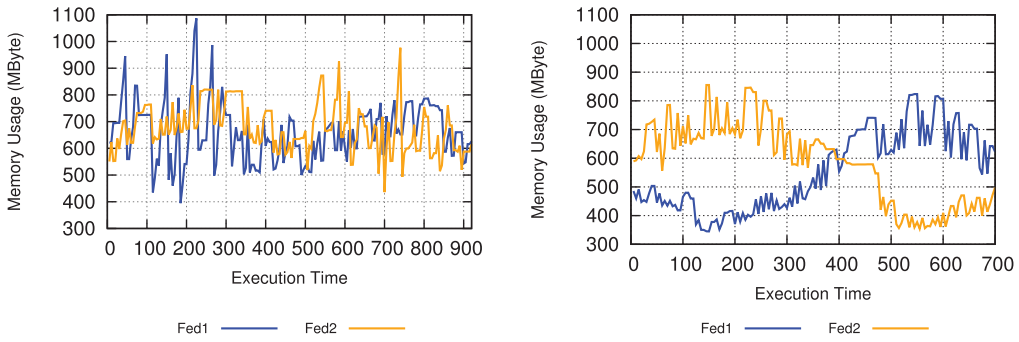
(b) Execution speed v.s. $P_{ExternEvent}$



(c) Execution speed v.s. LA

Fig. 9. Performance comparison between naive and ARMA models.

Finally, a performance comparison is made between the naive and ARMA models employed in the resource manager of ArmVee. Similarly, we report the experimental results for the Case (iii) simulation workload only. Figure 9(a) shows the simulation execution speed with respect to the decreasing control interval length when $P_{ExternEvent} = 1\%$ and $LA = 45$ (The ARMA curve in the figure is the same as the A(iii)



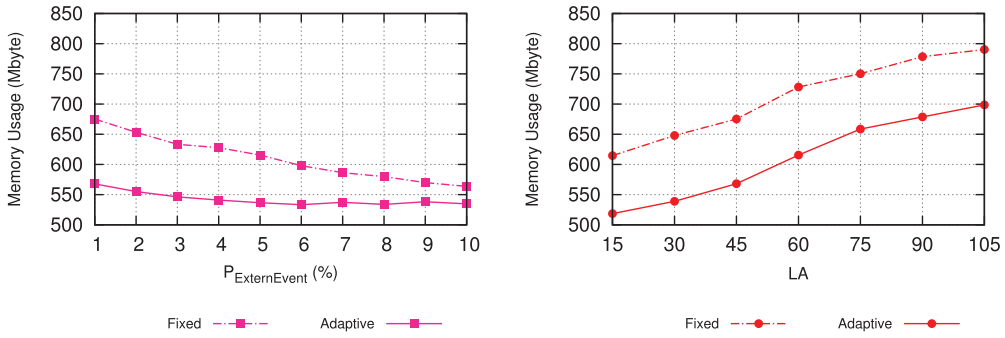
(a) Memory trace in fixed method (b) Memory trace in adaptive method

Fig. 10. Trace of memory usage in Fixed and Adaptive methods.

curve in Figure 6(a)). As we can see, the ARMA model can always outperform the naive model. The larger the control interval length, the more significant the performance advantage. This is because the naive model cannot accurately predict the simulation workload, especially when the control interval length is large. In contrast, the ARMA model can capture the relationship between VM cap and federate execution speed accurately in the presence of dynamic simulation workload. Figure 9(b) shows the simulation execution speed with respect to the increasing $P_{ExternEvent}$ when $LA = 45$. (The ARMA curve in the figure is the same as the *A-45* curve in Figure 7(a)). Figure 9(c) shows the execution speed with respect to the increasing LA when $P_{ExternEvent} = 1\%$. As we can see, the ARMA model can always outperform the naive model. This further verifies that the ARMA model is more accurate than the naive model.

6.2.2. Reducing Memory Usage. As analyzed in Section 5.2, ArmVee is able to reduce the memory usage as well. In our experiment, we trace the memory usage of VMs every 5s. The traces are illustrated in Figure 10, for the situation in which $LA = 45$ and $P_{ExternEvent} = 1$. As we can see, the memory usage in both *Fixed* and *Adaptive* methods changes very frequently. This is because optimistic federates save their execution states and conduct fossil collections during the simulation execution. In the *Fixed* method, the federate with lower simulation workload has a smaller sized execution state. However, it probably executes faster, thus has to save a greater number of execution states. For this reason, the memory usage (Figure 10(a)) does not change corresponding to the dynamic workload (Figure 4(c)). In the *Adaptive* method, federates have comparable execution speed. Hence, they have a similar number of saved execution states. Hence, the memory usage in federates changes according to the size of their execution states, which are determined by the simulation workload. For this reason, we can observe similar trends between Figure 4(c) and Figure 10(b). Last, but not least, we can observe from Figure 10 that the memory usage in the *Adaptive* method is smaller than that in the *Fixed* method.

Figure 11(a) compares the averaged memory usage of federates with respect to the increasing $P_{ExternEvent}$ when $LA = 45$. The averaged memory usage in the *Adaptive* method is almost constant regardless of the value of $P_{ExternEvent}$. In contrast, the averaged memory usage in the *Fixed* method decreases with the increasing $P_{ExternEvent}$. This is because the number of execution rollbacks increases (see Figure 7(b)) and each execution rollback reclaims those saved execution states during the overoptimistic execution. Figure 11(b) compares the averaged memory usage of federates with respect to the increasing LA when $P_{ExternEvent} = 1\%$. Generally, large LA will result in a large difference of simulation times among federates. Thus, federates have a greater number



(a) Memory usage v.s. $P_{ExternEvent}$ (b) Memory usage v.s. LA

Fig. 11. Averaged memory usage in Fixed and Adaptive methods.

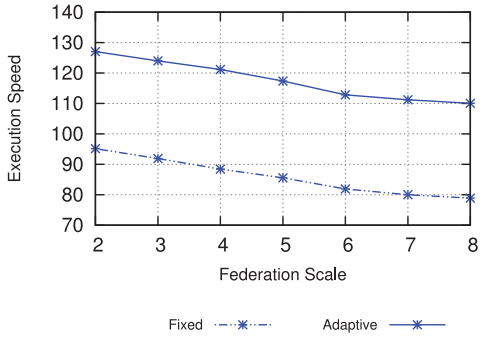
of saved states in case of future execution rollbacks. Therefore, we can observe that the memory usage in both *Fixed* and *Adaptive* methods increases with the increasing *LA*. Finally, we can observe from Figures 11(a) and 11(b) that the averaged memory usage in the *Adaptive* method is always smaller than that in the *Fixed* method.

6.2.3. Investigating Scalability. In order to study the scalability of ArmVee, the federation scale N is increased from 2 to 8. The simulation workload of each federate is obtained from the trace of RuneScape [Nae et al. 2008]. The time difference between the data centers simulated by two subsequent federates is set to $24/N$ hours. During the simulation, a federate may send a message to one of the other federates to simulate the communication (e.g., game state replication) between their corresponding data centers. The total CPU resource for the federation execution increases with the federation scale. Specifically, the number of CPU cores M is equal to $N/2$, thus $CAP = 100 \times M = 50 \times N$. Due to space limitations, we show only the experimental results of the situation in which $LA = 105$ and $P_{ExternEvent} = 1$. Figure 12 illustrates the simulation execution speed, number of execution rollbacks, and execution efficiency with the increasing federation scale. Compared with the *Fixed* method, the *Adaptive* method encounters less execution rollbacks, has higher efficiency, and thus has much greater execution speed. In both *Fixed* and *Adaptive* methods, the execution speed decreases slightly with the increasing federation scale. Hence, we can claim that ArmVee achieves significant execution speedup and has good scalability.

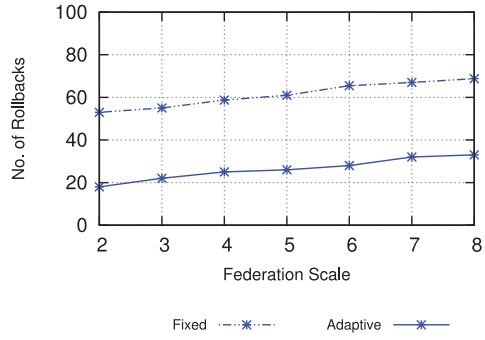
6.2.4. Verifying Theoretical Analysis. Finally, to verify the theoretical analysis in Section 5.1 on the execution speedup of ArmVee, synthetic workload is introduced to the MMOGs simulation model. The number of players connected to each data center follows a normal distribution $N(\mu, \sigma^2)$. First, we consider two federates in the federation. In the first federate (F_1), $\mu = 300$, $\sigma = 5$; in the second federate (F_2), $\mu = 300 \times W$, $\sigma = 5 \times W$, where W increases from 1 to 8. Since the event processing time increases proportionally with the number of players, W is the ratio of simulation workload of F_2 versus that of F_1. In the case that two federates are allocated with the same CPU resource, we get $MES_1 = W \times MES_2$. According to the analysis in Section 5.1, we get $\omega_2 = 1$ and $\omega_1 = W$. Hence, in theory, the execution speedup of ArmVee is:

$$\frac{N}{\sum \frac{1}{\omega_i}} \approx \frac{2W}{1+W}. \tag{11}$$

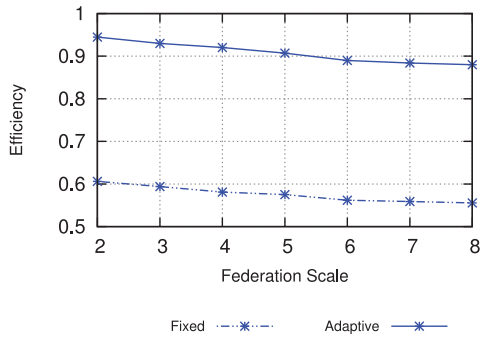
As shown in Figure 7, in the situation in which $LA = 105$ and $P_{ExternEvent} = 1$, federates in the *Adaptive* method encounter only a few execution rollbacks, and the execution



(a) Execution Speed

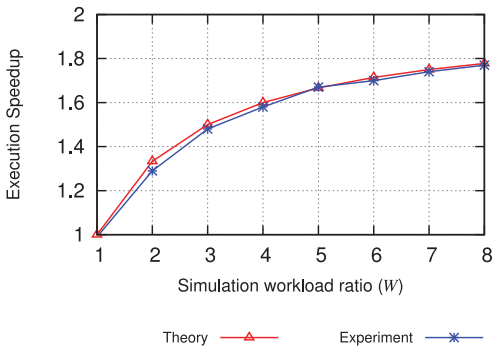


(b) Execution Rollbacks

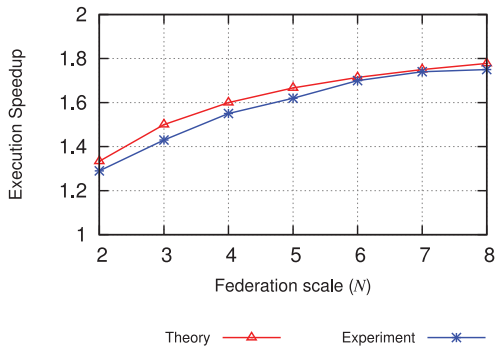


(c) Execution Efficiency

Fig. 12. Performance comparison between Fixed and Adaptive methods with the increasing federation scale.



(a) Execution speedup v.s. federate workload ratio



(b) Execution speedup v.s. federation scale

Fig. 13. Execution speedup of Adaptive method over Fixed method in theory and experimental conditions.

efficiency is close to a value of one. Therefore, the execution speedup of ArmVee in the experiment are very close to those values calculated in theory, as shown in Figure 13(a).

In addition, experiments are also carried out using the synthetic workload with federation scale (N) increased from 2 to 8. For F_i, $\mu = 300 \times i$, $\sigma = 5 \times i$. Similar to Section 6.2.3, a federate may send a message to one of the other federates and $CAP = 50 \times N$. According to the analysis in Section 5.1, we get $\omega_N = 1$ and $\omega_i \approx N/i$.

Hence, in theory, the execution speedup of ArmVee is:

$$\frac{N}{\sum \frac{1}{\omega_i}} \approx \frac{2N}{1+N}. \tag{12}$$

Similarly, as shown in Figure 13(b), the execution speedup of ArmVee in the experiment is also close to those values calculated in theory in the situation in which $LA = 105$ and $P_{ExternEvent} = 1$.

7. CONCLUSION AND FUTURE WORK

In this article, we have proposed an Adaptive Resource provisioning Mechanism in Virtual Execution Environment (ArmVee) for the purpose of improving performance of optimistic parallel and distributed simulation. It is composed of a performance monitor and a resource manager. The performance monitor, as a middleware, measures the performance of individual federates transparently to the simulation application. The resource manager adopts a prediction model used in control theory to capture the relationship between the performance of a federate and the resource share (i.e., cap value) of the corresponding VM. Therefore, it can distribute the available resources among federates, while ensuring that they can advance their simulation time with comparable speeds. Consequently, ArmVee can avoid wasting resources on overoptimistic executions and execution rollbacks, and the execution time of the whole federation can be reduced. We have also analyzed theoretically the advantages of our proposed ArmVee in both accelerating simulation execution and reducing memory usage. In order to evaluate the ArmVee, experiments are carried out using a MMOGs ecosystem simulation model in which federates are likely to have dynamic and imbalanced simulation workloads. Experimental results have shown that, in most of the investigated cases, ArmVee can speed up the simulation executions and reduce memory usages significantly. Experiments have also verified that ArmVee is scalable and that it is able to achieve execution speedup close to the values calculated in our theoretical analysis.

In the future, we will extend ArmVee to support multithreaded federates that are composed of a group of concurrent and tightly dependent threads [Jagtap et al. 2012]. Co-scheduling solutions [Weng et al. 2011; Sukwong and Kim 2011] in which 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. We will also integrate our CPU resource controller with a memory controller [Heo et al. 2009], disk I/O controller [Padala et al. 2009], and network controller [Popa et al. 2011] to develop a comprehensive solution to speed up simulation executions using minimum resources. Furthermore, ArmVee will be applied for various parallel and distributed applications on large-scale execution environments (e.g., data centers and the Cloud), taking workload balance and fault tolerance issues into account.

REFERENCES

P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. 2003. Xen and the art of virtualization. *SIGOPS Operating Systems Review* 37, 5, 164–177.

S. K. Barker and P. Shenoy. 2010. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of Conference on Multimedia Systems (MMSys'10)*. 35–46.

L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. 2006. An adaptive load balancing middleware for distributed simulation. In *Proceedings of the 2006 International Conference on Frontiers of High Performance Computing and Networking (ISPA'06)*. 873–883.

R. E. Bryant. 1977. Simulation of packet communication architecture computer systems. Technical Report. Massachusetts Institute of Technology. Cambridge, MA.

C. D. Carothers, D. Bauer, and S. Pearce. 2000. ROSS: A high-performance, low memory, modular time warp system. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS'00)*. 53–60.

K. M. Chandy and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering* 5, 5, 440–452.

L. Chen, Y. Lu, Y. Yao, S. Peng, and L. Wu. 2011. A well-balanced time warp system on multi-core environments. In *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS'11)*. 1–9.

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

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

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

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

T. N. B. Duong, X. Li, R. S. M. Goh, X. Tang, and W. Cai. 2012. QoS-aware revenue-cost optimization for latency-sensitive services in IaaS clouds. In *Proceedings of the 16th International Symposium on Distributed Simulation and Real Time Applications (DS-RT'12)*. 11–18.

R. M. Fujimoto. 2000. *Parallel and Distributed Simulation Systems*. Wiley Interscience, New York, NY.

R. M. Fujimoto, A. W. Malik, and A. J. Park. 2010. Parallel and distributed simulation in the cloud. *SCS Modeling and Simulation Magazine, Society for Modeling and Simulation, Intl.* 1 (July 2010). Issue 3.

Z. Gong, X. Gu, and J. Wilkes. 2010. PRESS: PRedictive elastic ReSource scaling for cloud systems. In *Proceedings of International Conference on Network and Service Management (CNSM'10)*. 9–16.

J. E. Hannay, K. Bråthen, O. M. Mevassvik, and A. Skjeltorp. 2014. Live, virtual, constructive (LVC) simulation for land training: Concept development & experimentation (CD&E). In *NATO Modeling and Simulation Group Symposium on Integrating Modelling & Simulation in the Defence Acquisition Lifecycle and Military Training Curriculum*.

J. Heo, X. Zhu, P. Padala, and Z. Wang. 2009. Memory overbooking and dynamic control of Xen virtual machines in consolidated environments. In *Proceedings of the 11th International Symposium on Integrated Network Management (IM'09)*. 630–637.

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

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

D. R. Jefferson. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems* 7, 3, 404–425.

E. Kalyvianaki, T. Charalambous, and S. Hand. 2009. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In *Proceedings of International Conference on Autonomic Computing (ICAC'09)*. 117–126.

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

Z. Li, X. Li, T. N. B. Duong, W. Cai, and S. J. Turner. 2013. Accelerating optimistic HLA-based simulations in virtual execution environments. In *Proceedings of ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS'13)*.

Y. Lin, B. R. Preiss, W. M. Loucks, and E. D. Lazowska. 1993. Selecting the checkpoint interval in Time Warp simulation. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS'93)*. 3–10.

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

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

J. Mason. 2009. A Detailed Look at Data Replication Options for Disaster Recovery Planning. White Paper.

V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D. Epema, and T. Fahringer. 2008. Efficient management of data center resources for massively multiplayer online games. In *Proceedings of the Conference on Supercomputing (SC'08)*. 10:1–10:12.

P. Padala, K. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. 2009. Automated control of multiple virtualized resources. In *Proceedings of European Conference on Computer Systems (EuroSys'09)*. 13–26.

K. Pan, S. J. Turner, W. Cai, and Z. Li. 2007. A service oriented HLA RTI on the grid. In *Proceedings of Conference on Web Services (ICWS'07)*. 984–992.

K. S. Panesar and R. M. Fujimoto. 1997. Adaptive flow control in time warp. In *Proceedings of Workshop on Parallel and Distributed Simulation (PADS'97)*. 108–115.

L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. 2011. FairCloud: Sharing the network in cloud computing. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets'11)*. 22:1–22:6.

F. Quaglia. 2006. A middleware level active replication manager for high performance HLA-based simulations on SMP systems. In *Proceedings of 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'06)*. 219–226.

D. Schanzenbach and H. Casanova. 2008. *Accuracy and Responsiveness of CPU Sharing Using Xens Cap Values*. Technical Report. Computer and Information Sciences Department, University of Hawai at Manoa.

Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. 2011. CloudScale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of Symposium on Cloud Computing (SOCC'11)*. 5:1–5:14.

L. M. Sokol, D. P. Briscoe, and A. P. Wieland. 1988. MTW: A strategy for scheduling discrete simulation events for concurrent execution. In *Proceedings of the SCS Multiconference on Distributed Simulation*. 34–42.

W. Suh, M. P. Hunter, and R. Fujimoto. 2014. Ad hoc distributed simulation for transportation system monitoring and near-term prediction. *Simulation Modelling Practice and Theory* 41, 1–14.

O. Sukwong and H. S. Kim. 2011. Is co-scheduling too expensive for SMP VMs? In *Proceedings of the Sixth Conference on Computer Systems (EuroSys'11)*. 257–272.

R. Vitali, A. Pellegrini, and F. Quaglia. 2012. Towards symmetric multi-threaded optimistic simulation kernels. In *Proceedings of the 26th Workshop on Principles of Advanced and Distributed Simulation (PADS'12)*. 211–220.

X. Wang, S. J. Turner, M. Y. H. Low, and B. P. Gan. 2005. Optimistic synchronization in HLA-based distributed simulation. *Simulation* 81, 4, 279–291.

C. Weng, Q. Liu, L. Yu, and M. Li. 2011. Dynamic adaptive scheduling for virtual machines. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC'11)*. 239–250.

D. Wesam, T. Ibrahim, and M. Christoph. 2012. Elastic virtual machine for fine-grained cloud resource provisioning. *Communications in Computer and Information Science Volume* 269, pp 11–25.

Xen. 2013. Xen Credit Scheduler. Retrieved May 25, 2015 from http://wiki.xen.org/wiki/Credit_Scheduler.

Y. Yang, D. Shang, and J. Huang. 2012. Fixed, spot or flexi pricing: An integrated prototype for alternate cloud computing pricing mechanisms. In *Proceedings of 22nd Annual Workshop on Information Technologies and Systems (WITS'12)*.

S. Yoginath and K. Perumalla. 2013. Optimized hypervisor scheduler for parallel discrete event simulations on virtual machine platforms. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques (SIMUTOOLS'13)*.

Z. Yuan, W. Cai, Y. Low, and S. J. Turner. 2004. Federate migration in HLA-based simulation. In *Proceedings of Conference on Computational Science*. 856–864.

Received November 2013; revised October 2014; accepted January 2015