A Multi-layered Scheme for Distributed Simulations on the Cloud Environment

by

Shichao Guan

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the M.A.Sc. degree in

Electrical and Computer Engineering

School of Electrical Engineering and Computer Science

Faculty of Engineering

University of Ottawa

Abstract

In order to improve simulation performance and integrate simulation resources among geographically distributed locations, the concept of distributed simulation is proposed. Several types of distributed simulation standards, such as DIS and HLA are established to formalize simulations and achieve reusability and interoperability of simulation components. In order to implement these distributed simulation standards and manage the underlying system of distributed simulation applications, Grid Computing and Cloud Computing technologies are employed to tackle the details of operation, configuration, and maintenance of the simulation platforms in which simulation applications are deployed. However, for modelers who may not be familiar with the management of distributed systems, it is challenging to create a simulation-run-ready environment that incorporates different types of computing resources and network environments. In this thesis, we propose a new multi-layered cloud-based scheme for enabling modeling and simulation based on different distributed simulation standards. The scheme is designed to ease the management of underlying resources and achieve rapid elasticity, providing unlimited computing capability to end users; energy consumption, security, multi-user availability, scalability and deployment issues are all considered. We describe a mechanism for handling diverse network environments. With its adoption, idle public resources can easily be configured as additional computing capabilities for the local resource pool. A fast deployment model is built to relieve the migration and installation process of this platform. An energy conservation strategy is utilized to reduce the energy consumption of computing resources. Security components are also implemented to protect sensitive information and block malicious attacks in the cloud. In the experiments, the proposed scheme is compared with its corresponding grid computing platform; the cloud computing platform achieves a similar performance, but incorporates many of the cloud's advantages.

Acknowledgements

I am very grateful to Prof. Azzedine Boukerche, who has offered me the precious opportunity to study in the Paradise Lab, provided all the equipment that necessary for my experiments, offered me RA funding to support my study, and showed me a way through the mysterious scientific world. Like a mariner, I can chart my course by his lights.

I am also very appreciative of the kind support from Dr. Robson, who has always illuminated me throughout my graduate period. His rigorous studying attitude and passion encourages me to keep moving forward.

I thank all the members in the Paradise Lab. Thank you for all your help.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 2
- 1.3 Contribution . . . 3
- 1.4 Outline . . . 3

2 Background **5**
- 2.1 Distributed Simulations . . . 6
- 2.2 Distributed Simulation Standards . . . 7
  - 2.2.1 HLA Protocol . . . 7
  - 2.2.2 DIS Protocol . . . 9
  - 2.2.3 DDS Protocol . . . 10
- 2.3 Grid Computing and Cloud Computing . . . 12

3 Related Work **16**
- 3.1 Grid-based Distributed Systems . . . 16
  - 3.1.1 HLAGrid . . . 17
  - 3.1.2 HLA-GRID-REPAST . . . 18
  - 3.1.3 A Framework for HLA-Based Interactive Simulations on the Grid 18
  - 3.1.4 Load Management System for HLA-based Distributed Simulations on the Grid . . . 19
  - 3.1.5 Dynamic Load Balancing Using Grid Services for HLA-Based Simulations on Large-Scale Distributed Systems . . . 20
- 3.2 Cloud-based Distributed Systems . . . 21
  - 3.2.1 MapReduce . . . 21
  - 3.2.2 Aurora System . . . 22
  - 3.2.3 TW-SMIP . . . 23
  - 3.2.4 CSim . . . 24
  - 3.2.5 CDS . . . 24
  - 3.2.6 RACS . . . 25
  - 3.2.7 Summary . . . 27

4 The Cloud Simulation Platform **30**
4.1 Deployment Management Layer 32
4.1.1 Simulation Resource Deployer 32
4.1.2 Virtual Resource Deployer 33
4.1.3 Cloud Infrastructure Deployer 34
4.1.4 Cloud Simulation Platform Packaging and Installation Algorithm 34
4.2 User Management Layer 37
4.2.1 Authentication and Access Control Algorithm 38
4.3 Simulation Function Layer 39
4.3.1 Distributed Simulation Workbench 39
4.3.2 Simulation Resource Manager 41
4.3.3 Virtual Resource Manager 42
4.3.4 Scheduling Algorithm and Dynamic VM Reallocation Algorithm . 43
4.4 Ingegration and Virtualization Layer 45
4.4.1 Virtual Resource Manager 46
4.4.2 Public Resource Agent 47
4.4.3 Simulation Resource Provisioning and Mapping Algorithm 48
4.5 Raw Resource Layer 50

5 Experiment Results and Discussion **51**
5.1 Experiment Environments 51
5.2 Performance Analysis of Environment Preparation 53
5.3 Performance Analysis of Job Scheduling 55
5.4 Analysis on Energy Consumption 57
5.5 Analysis on Simulation Performance 59
5.6 Summary 61

6 Conclusion and Future Work **62**
6.1 Conclusion 62
6.2 Future Work 63

List of Tables

2.1 PDUs in DIS 11

3.1 Comparisons in Cloud Simulation Platforms 1 28
3.2 Comparisons in Cloud Simulation Platforms 2 29

5.1 Single Machine Environment 52
5.2 Virtual Resource List 54
5.3 Computing Resource Preparation in the Single Machine Mode 55
5.4 Computing Resource Preparation in the Cluster Mode 55
5.5 Packages Handling Process 56
5.6 CloudSim Simulation Parameters 58
5.7 Scheduling with Migration 59
5.8 Scheduling without Migration 59

List of Figures

2.1 HLA Framework 8

3.1 MapReduce Execution Process 22
3.2 Aurora Framework 23
3.3 Cloud-RTI Framework 25
3.4 Cloud Simulation System Framework 26

4.1 Cloud Simulation Scheme 31

5.1 Scheduling Algorithm Performance Evaluation 56
5.2 Simulation Performance between Cloud Platform and Grid Platform . . . 60

Chapter 1

Introduction

1.1 Motivation

Due to the benefits of the Cloud Computing, such as the flexible provision of resources, rapid scaling capability, and utilization of virtualization technology, many Cloud providers offer the public cloud services such as AWS [58], Google AppEngine [78], Microsoft Azure [87] and IBM Cloud [89], on a pay-per-use basis, known as utility cloud, which boosts the development of the Cloud Computing in the industry area. Public cloud providers offer geographic tools for users to manage their cloud instances, embedding usage measurement elements for charging. Meanwhile, the concept of private cloud is introduced as a new approach to manage small data centers; this allows computing resources, storage resources and network resources to be integrated and virtualized for centralized management. Although this method is a very promising for the provision resources, it is not

tailored to suit the requirements of distributed simulations such as inter-cloud usability, network latency tolerance, or the support of communication messages.

For large-scale distributed simulations, the management and maintenance of underlying resources involves a high degree of complexity. Many efforts have been made to build a simulation-run-ready environment. In addition, for shared resources in the simulations, the security issues should be carefully concerned. As the scale of the simulation increases, the energy consumption of the resources demands more attention.

Some cloud platforms proposed for the distributed simulations are discussed in detail in the next chapter; however, they lack the concerns of usability, energy consumption, security, comparison study or inter-cloud issues.

1.2 Objectives

As discussed in the previous section, it is feasible to design a PaaS (Platform as a Service) for the distributed simulations based on the characteristics of distributed simulation and cloud computing. Such a PaaS should satisfy the requirements of distributed simulation standards, and support the management and maintenance of underlying resources that these standards do not cover. The platform should enable automatic simulation environment deployment, consider energy consumption, and provide user-friendly UI to manage underlying resources. For the simulation itself, job scheduling algorithms should be designed and fault tolerance should be achieved. Also, the platform should be portable and flexible enough to support inter-cloud communication among geographically distributed

resources.

1.3 Contribution

In my thesis, I propose a multi-layered elastic cloud simulation scheme which supports the capability of automatic deployment, reduces energy consumption, and facilitates outward scaling. A deployment management scheme is proposed to tackle deployment and migration aspects of the cloud simulation scheme, providing a flexible and agile approach to replicate that function as backups. The energy consumption issue is addressed and solutions are proposed and implemented. The scaling process is also implemented; it responds rapidly to requirements from cloud users, providing unlimited computing resources to end users. The scaling components support idle public instances such as physical desktops, workstations, laptops, or public cloud instances from different cloud providers (AWS, Google App Engine, or Azure) which can be located in geographically separate regions.

1.4 Outline

The remainder of this thesis is organized as follows. In Chapter 2, general background knowledge regarding the proposed system is introduced. In Chapter 3, state-of-the-art related research work is shown and discussed. In Chapter 5, the proposed scheme is described, introducing architecture, key components, functioning and implementation.

In Chapter 6, several experiments are presented and discussed in terms of energy consumption, performance, and results. To conclude, the proposed solution design and experimental results are summarized, and direction for future works is provided.

Chapter 2

Background

Computer simulations were facilitated to design, create, and evaluate complex real systems on computers. Compared to traditional simulations based on mathematical models, computer simulations with computing models achieve better observation, analysis and understanding of the behavior of the real process. In the 1970s and 1980s, as shown in [53], the parallel and distributed simulation in the computer simulation area attracted considerable attention and its maturity increased due to its benefits, such as execution efficiency and interoperability. Since then, many standards have been proposed to formalize the concept of distributed simulations. Meanwhile, with growing scale of computing simulation systems, issues pertaining to management and maintenance increasingly complex underlying physical resources have received great interest. To accomodate this issue, many Grid Computing schemes and Cloud Computing schemes have been proposed for the management and maintenance of distributed simulation systems.

2.1 Distributed Simulations

Parallel simulations and distributed simulations refer to technologies that enable the execution of a simulation within a computing system containing multiple processors [53]. Compared to a traditional computer simulation, the distributed simulation introduces several new characteristics.

In the distributed simulations, a collection of different simulators execute tasks simultaneously, communicating with each other during run-time. In this manner, the execution of simulations is sped up, and the length of execution time is reduced compared to conventional sequential simulations. Unlike sequential simulations, which contain an event list including all pending tasks and a global clock that denotes how far the simulation has progressed, the distributed simulations must guarantee that events are executed in chronological order and avoid dead locks, due to the lack of an overall global clock.

In addition to the advantages mentioned above, multiple participants from geographical distributed locations can be embedded in the distributed virtual environments (DVEs). In this case, distributed participants are able to communicate with each other in the virtual local environment. In addition, coordination mechanisms are provided to support the integration of simulators from different organizations, reducing the costs that may introduced by manually merging the geographically distributed simulators together.

2.2 Distributed Simulation Standards

Several approaches have been designed for distributed simulation systems, such as the Distributed Interactive Simulation (DIS) [2], its successor High Level Architecture (HLA) [80], and the Data Distribution Service for Real-Time Systems (DDS) [67]. The DIS is mainly designed for military simulations, and introduces design concepts such as interoperability, autonomy, and dead reckoning that cater to the development of distributed commercial applications. The HLA, initially developed by the Department of Defence in the United States, is an widely-used framework that provides reusability and interoperability for distributed simulations. By adopting this framework, modeling and simulations have been enabled for supporting analysis, engineering, military, entertainment, education, and a variety of other applications that can be linked to live systems (collectively defined as federates). The DIS and the HLA have been also developed as IEEE standards for modeling and simulation. The DDS, managed by the Object Management Group, is used as a messaging middleware standard that supports data-centric simulations, enabling seamless, timely, scalable, and dependable distributed data sharing.

2.2.1 HLA Protocol

High Level Architecture, defined in [80] and [41], is a general framework facilitating interoperability and reusability of distributed simulation components, reducing the risk and expense of re-implementing or retrofitting already existing simulators. Guided by this standard, a component simulation may be used in different scenarios and applications

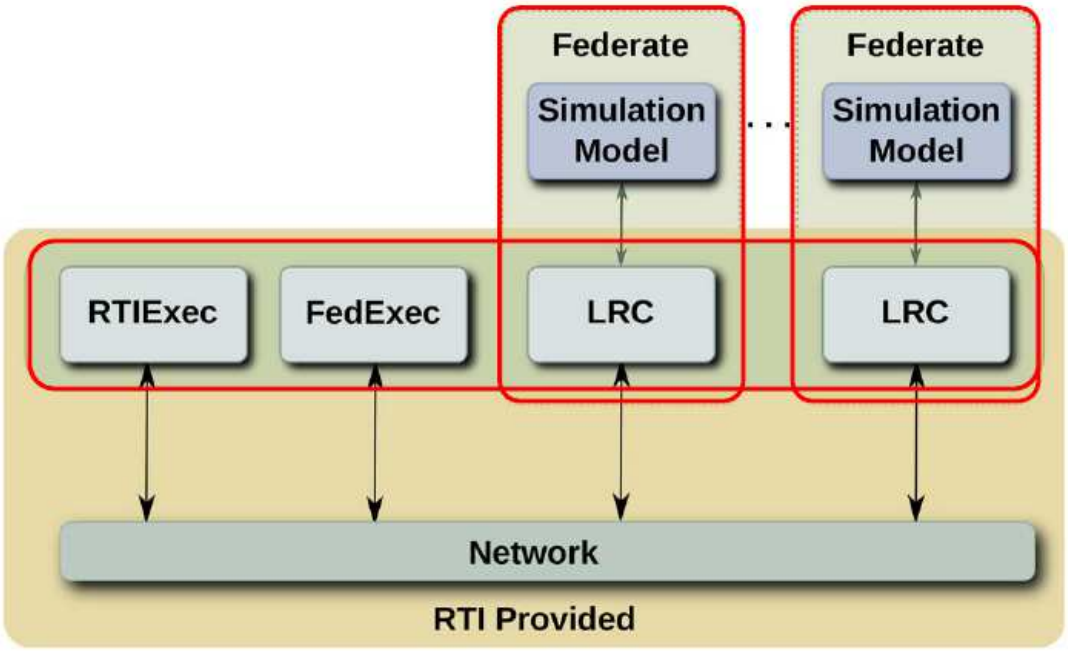

Figure 2.1: HLA Framework

over its lifetime; simulations distributed across heterogeneous hardware and software platforms can be aggregated; and components can be reused without significant code change or development cost.

HLA components include the HLA rules, Interface Specification, and Object Model Template (OMT). The HLA rules describe the simulation and federate responsibilities, ensuring the proper interaction of simulations in a federation. The Interface Specification defines Run-Time Infrastructure (RTI) services and identifies "callback" functions for each federate. The OMT establishes the format of key models, such as Federation Object Model (FOM), Simulation Object Model (SOM), and Management Object Model (MOM). In addition, it provides a common method for recording information.

The execution process is depicted in 2.2.1. The RTIExec manages the creation and destruction of multiple federation executions. The FedExec is a runtime instantiation of a federation and manages multiple federates joining and leaving the federation execution. The Local Run-time Component (LRC) is for federates to access for the use of management services. The federate is an application that supports the HLA, and is capable of participating in a simulation. When a federation is run, the RTIExec is started first. Then a federate, acting as a manager, creates a federation execution by invoking the RTI method "createFederationExecution" on its RTI Ambassador. The RTI Ambassador then reserves a name with RTIExec which spawns a FedExec process and that FedExec registers its communication address with RTIExec. Once a federation execution exists, other federates can join it. That RTI Ambassador consults RTIExec to obtain the address of FedExec, and invokes "joinFederationExecution" on FedExec. Additional federates can join via the same process.

Middlewares such as poRTIco [72], Pitch [73], CISUC [7], and SimWare [79] are implemented based on the requirements of HLA standards.

2.2.2 DIS Protocol

Originating in the SIMNET project shown in [65], the Distributed Interactive Simulation, defined by [2] and [57], is proposed as a standardized approach for message changing in a virtual world. It provides common semantics for coordinating systems, describing specific entities, tackling issues such as message specification, coordination of simulation

components, network communication, scalability, interoperability and latency. Based on the concept of DIS, the simulation exercise does not require central nodes to control the entire process; the state of simulation entities and objects are changed and maintained by the control of individual applications, and dead reckoning is used to reduce the amount of communication.

In order to support interoperability among different simulators, standard Protocol Data Units (PDUs) are proposed to support run-time communication. 72 different PDUs are arranged into 13 families, as shown in Table 2.1, according to the DIS standard. Software such as OpenDIS [1] and KDIS [59] are implemented that support the DIS standard.

2.2.3 DDS Protocol

To provide efficiency and convenience for messaging among diverse simulators, the concept of data distribution is proposed in order to tackle communication issues in the underlying network, which defines mechanisms for describing message transmission. In Data Distribution Service, explained in [67] and [70], two layers of interfaces are defined: the Data-Centric Publish-Subscribe (DCPS), which focuses on information delivery, and the optional Data Local Reconstruction Layer (DLRL), which concerns service integration.

The DCPS provides approaches that enable features for the publishing and subscription process, which includes the Platform Independent Model (PIM) and the Plat-

Table 2.1: PDUs in DIS

PDU Family	Key Features
Entity information/interaction	Appearance, Collision, Entity State, Attribute
Warfare	Fire, Directed Energy, Entity Damage Status
Logistics	Request, Response, Resupply, Repair
Simulation Management	Start, Stop, Resume, Data Collection
Distributed Emission Regeneration	Emissions
Radio Communications	Transmitter, Signal, Receiver, Intercom Signal, Intercom Control
Entity Management	Aggregation, Ownership Changes
Minefield	Location, Appearance
Synthetic Environment	Information Exchange
Simulation Management with Reliability	Reliable Communications
Information Operations	Predicted Effects, Decision-Making
Non-Real Time (NRT) protocol	NRT Support
Live Entity	Bandwidth

form Specific Model (PSM). Here, five types of models are developed; these support notification-based and wait-based interaction, provide the factory for classes and the container for objects, and manage QoS rules. In addition, a mapping mechanism is set for PIM and PSM, translating UML interfaces and classes into IDL interfaces. DLRL is an optional layer that attempts to ease the use of DCPS, and supports object orientation.

Coordinated by the aforementioned standards and frameworks, large-scale distributed simulations can be deployed to geographically distributed computing resources. However, within the specifications of the standards and frameworks, mechanisms for executing simulations are not provided for the underlying systems. For each element of physical computation, it is necessary to carefully perform a series of configurations to build the simulation-run-ready environments so that simulations can be properly executed. Software platforms such as OpenDDS [68] and SpliceDDS [74] support the DDS requirements.

2.3 Grid Computing and Cloud Computing

In order to meet the need for handling the underlying system, Grid Computing, as described in [50], [51] and [11], is introduced to manage shared resources and the scheduling distributed simulations. Based on the definition of Grid Computing in the above articles, the Grid should coordinate resources that are not subject to centralized control; use standard, open, general-purpose protocols and interfaces; deliver non-trivial qualities of service. For distributed simulations, a Grid system can perform as a coordinated resource sharing system that provides services such as security, resource management, informa-

tion queueing, and data management for distributed applications. The Globus Toolkit (GT) [3] is defined as a de-facto middleware standard for Grid Computing that helps attain seamless interoperability. Grid-based simulation platforms, compared to traditional computing simulation systems, overcome limitations in terms of resource distribution, dynamic access, security, organization, and collaboration with the help of Grid services [49].

Even though the Grid eases the management of underlying systems for distributed simulations, there are still problems with its ability to handle resources in a fine-grained manner. As a result, Grid-based platforms cannot reallocate resources during runtime while fully supporting multi-users. To tackle these issues with the underlying system, Cloud Computing (as introduced by [6], [69] and [35]) is employed as a new approach for managing underlying resources for distributed simulations.

Similar to Grid Computing, Cloud Computing provides more advantages according to the analyses in [52], [85], [36], and [86]. There are several reasons to migrate distributed simulations from the Grid to the Cloud:

- *Resource sharing*: Cloud provides resources on demand during runtime, while Grid emphasizes a fair share of resources across organizations.

- *Virtualization*: For Grid, virtualization mainly covers the softlayer, which concerns data and programming. For Cloud, in addition to the softlayer, virtualization covers hardware resources. By enabling abstraction and encapsulation of raw physical resources, the Cloud can provide better isolation and manageability for fine-grained

resources.

- *Scalability*: The Grid scales primarily by increasing the number of working nodes. The Cloud offers automatic resizing of virtualized hardware.

- *Payment*: Grid services are typically billed using a fixed rate, while Cloud users have the option of a pay-per-use flexible payment model.

Although Cloud Computing has not been standardized to inner interfaces for accessing resources, which may lead to an inconvenient situation if computing resources are managed by different cloud service providers, it is still a promising approach for managing distributed systems and executing distributed simulations. Compared with these Grid services, the Cloud services, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) are more agile and flexible, resulting in a greater ease of control of granularity of services.

Due to the advantages mentioned above, many cloud simulation platforms have been proposed that incorporate many aspects in their design, such as job scheduling, monitoring, and security. However, these existing cloud platforms are implemented based on local resources. In this case, to end users, the capabilities available for provisioning are limited to the size of its local resource pool. This is not in accordance with the essential characteristic of elasticity in Cloud Computing, according to the definition from the National Institute of Standards and Technology (NIST) [69]. In addition, the issue related to energy consumption is not fully covered, particularly for distributed simulation

applications that require a relatively long period of time to execute. The migration and deployment model of the cloud simulation platform itself is also not fully discussed.

In this thesis, I propose a multi-layered cloud simulation scheme that considers the usability, elasticity, energy consumption, and fast deployment. Components are designed and implemented to provide unlimited computing resources to end users by coordinating public compute resources during runtime. Energy-aware elements are utilized to reduce the unnecessary energy cost from computing resources. A deployment model is designed to accelerate the migration and deployment process of the cloud simulation platform. In addition, the scheme contains functions for job scheduling, monitoring, and a friendly web-based graphic interface to ease the configuration, operation, and maintenance of the underlying system.

Chapter 3

Related Work

Due to the challenges in managing the underlying simulation environments and load of simulations (as mentioned in the previous chapter), the Grid Computing and Cloud Computing are extensively used to handle the details of physical resources, such as computing, storage and network resources.

3.1 Grid-based Distributed Systems

In order to facilitate the development of distributed simulations on the Grid systems (particularly HLA-based simulations), much effort has been made in the area of service provisioning, heterogeneity, resource distribution and load management.

3.1.1 HLAGrid

In the HLAGrid [88], a Federate-Proxy-RTI architecture is proposed which enables federates to be exposed as Grid services. In this case, simulation resources can be configured, maintained and scaled more easily, hiding the details of the underlying platforms. In this proposed scheme, a client-resource model is implemented, dividing the federate codes and management services. A Proxy is introduced to communicate at remote resources.

On the client side, the Globus [3] is used as the Grid middleware. The main function for the client side is to enable the communication of HLA RTI by means of the Grid channel, exposing the RTI as a grid service that can be called remotely. This feature is achieved by the author-defined proxy and callback function in the RTI. The resource side mainly contains the proxy, focusing on the translation of grid service and normal RTI services for the resource part.

For the execution process, the scheme adopts similar steps as a traditional HLA execution process. Due to its client-resource model, the RTI information should be properly stored and handled in the resource part, and method calls at the client part should be translated into the format of grid services, registering in the index services.

The results of the experiment showed that the proposed scheme introduced a huge latency compared to native HLA. This is due to the additional translation process during the execution. During the execution of different federates in the distributed simulations, the translation is necessary for communication between the federates.

3.1.2 HLA-GRID-REPAST

The HLA-GRID-REPAST [83] is a continuous work of HLAGrid, which coordinates the HLA_REPAST [40] and HLA_Grid. The HLA_REPAST middleware coordinates the HLA-GRID; it provides an agent-based solution that enables distributed simulations which require computing resources and data sets from geographically distributed locations. In the proposed scheme, HLA_REPAST is introduced to provide consistency and reliability for the communication of RTI. It also adopts a client-proxy model. Due to the existence of the translation process in this scheme, the simulation execution time of the experiment shows a higher latency when it is compared with native HLA_REPAST.

3.1.3 A Framework for HLA-Based Interactive Simulations on the Grid

Rycerz et. al [76] described a management framework designed for HLA-based simulations in the grid environments, which enables HLA Management Service, Migration Support Service and Broker Support Service. In the execution process, the Broker Client first starts the Broker Service, then installs the simulation application on the grid sites based on the HLA Speaking Services in the Registry Service list. Finally, the TRTExec is chosen based on the HLA Speaking Service. Taking load into consideration, the Benchmark Services and Monitoring Services are proposed to support decision-making by HLA Speaking Service. Based on the performance of local resources, simulation tasks can be migrated during execution to balance the computation load of the whole system. In

the migration scenario, a two-step migration is designed: first, it copies and instantiates the migration resources in the target site, then configures the migration resources with real time data. This may also lead to overhead due to the suspension of the simulation execution.

3.1.4 Load Management System for HLA-based Distributed Simulations on the Grid

As shown by Boukerche et al. [37], a two-layered structure is proposed, containing the job management sub-system and the resource management sub-system. The former is concerned with the monitoring and load balancing aspects, while the latter focuses on resource discovery and provisioning. The LMSHandler is implemented as a separate thread with HLA federates, and a corresponding interface is implemented. For the load balancing section, the scheme gathers load information in a specific interval – 5 minutes; this may not be the case for some applications. Also, this migration needs to suspend the execution process of the current simulation. In this system, the Graphical User Interface is implemented, easing the management of underlying resources and providing resource execution of simulation tasks in single log-in manner.

3.1.5 Dynamic Load Balancing Using Grid Services for HLA-Based Simulations on Large-Scale Distributed Systems

In the following work by Boukerche [34], a hierarchical dynamic load balancing scheme is designed to handle non-dedicated resources in the distributed system. Based on Grid services, this proposed scheme is able to detect, re-distribute, and migrate loads during the simulation execution time, improving simulation performance overall.

In the proposed scheme, the balancing tasks are divided into three distinct phases: Monitoring Phase, Re-distribution Phase and Migration Phase. In the Monitoring Phase, load-related data is collected and polished. Then, the resources are identified as overloaded, balanced or underloaded based on their current load. In the Re-distribution Phase, a pair-match is implemented between overloaded resources and underloaded resources to determine migration targets. In the Migration Phase, a two-step migration is designed to achieve data transfer.

Based on the experiments, this scheme improves the execution performance of distributed simulations, but does not consider communication load in runtime. In their following work [43], a distributed scheme is proposed that addresses the communication load between distributed resources. In [42], a predictive model is proposed. In this distributed dynamic scheme, communication load can be balanced by a proximity analysis of federate interactions. In [18], a conservative approach is proposed to balance the load during runtime. In [32], an efficient, scalable scheme is proposed for distributing state updates and interaction information in large-scale distributed simulations.

3.2 Cloud-based Distributed Systems

Cloud Computing offers many novel features and characteristics that the Grid cannot support, as mentioned in Section 1. As a result, many cloud-based simulation platforms are proposed for the distributed simulations, focusing on virtualization management, on-demand resource provisioning, security, monitoring, optimization of data processing, synchronization, etc.

3.2.1 MapReduce

The MapReduce [45] programming model is a widely used scheduling approach in the cloud environment. It formalizes division and parallel processes for computation tasks across multiple computing resources, achieving fault tolerance, locality optimization, and load balancing.

As depicted below, this model employs a master-worker pattern. The master handles the job assignment of splitting tasks and mapping, which effectively reduces workers, while the workers focus on the specific job execution. First, the job is divided into small sub-tasks, which are then distributed to map workers to execute by the master. The intermediate results from map workers are then mapped to reduce workers as directed by the master, and the reduce workers process these intermediate results accordingly, to produce the final output.

This approach facilitates cloud applications such as data mining, machine learning, and sorting. However, for distributed simulation applications, particularly those requir-

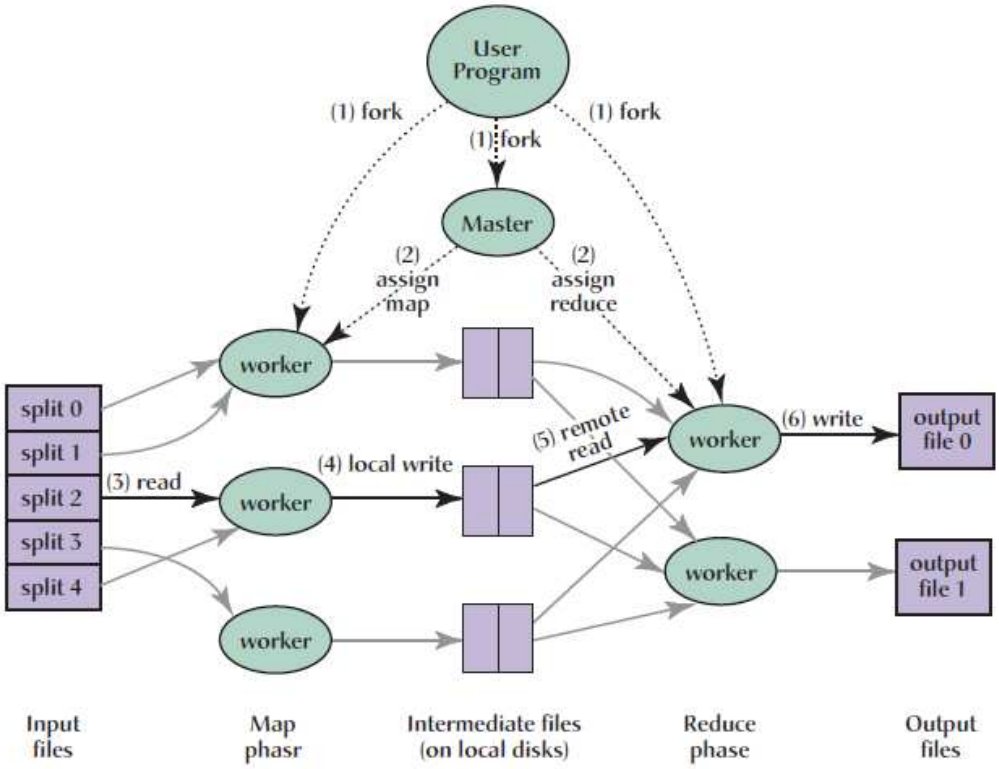

Figure 3.1: MapReduce Execution Process

ing intensive communications and frequent message exchange, this scheme cannot gain low enough latency to guarantee the performance of such simulations.

3.2.2 Aurora System

The Aurora system [54] also focuses on the task scheduling on the cloud environments. It is introduced to address the issue of small message communication in the cloud. In this system, a master/worker paradigm is implemented to aggregate small simulation communication messages, which are bundled and redirected to different destination resources. With the automatic bundling of small communication messages during the simulation

execution, the cloud's high bandwidth network can be better utilized.

As shown in the following figure, the system consists of the proxy, message state servers, work state servers and workers. During the execution process, the worker involves a data collection procedure, which packs the future messages for destination work units. Accordingly, the work units then download and unpack the assigned messages in a time-stamp order to process them.

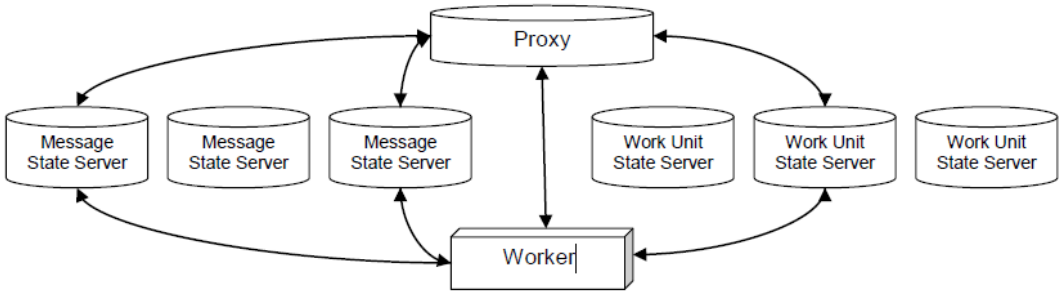

Figure 3.2: Aurora Framework

3.2.3 TW-SMIP

The Time Warp Straggler Message Identification Protocol (TW-SMIP) [64] is advanced to handle optimistic synchronization. In the TW-SMIP, heartbeat messages are used to detect straggler messages. The straggler message identification is utilized to reduce the number of rollbacks that may be caused by asymmetric or uneven processing loads. In addition, in this article, three types of SMIP protocols are proposed to distribute heart-beat messages that concern congestion, autonomous administration and communication overhead, respectively.

3.2.4 CSim

The Cloud-based Simulation (CSim) [63] concerns the idle CPU cycle issue. In this proposed scheme, each processor is virtualized as two CPUs (VCPU) – one foreground and one background. When the higher-priority foreground VCPU begins an idle cycle, the lower-priority, background VCPU begins working. According to the author, the foreground VCPU loses less than 4% performance due to this type of job scheduling; however, many idle CPU cycles can be used by the background VCPU. Based on this two-tier structure, four job scheduling algorithms are also proposed in this paper to improve the performance of parallel simulations on the cloud.

3.2.5 CDS

In the Cloud-based Distributed Simulation system (CDS) [56], as shown below, a new Cloud-RTI middleware based on traditional RTI is proposed. With the encapsulation of traditional RTI, this Cloud-based RTI is provided by the use of web services. In this protocol, optimization in the process of registration operations is also made by its Management Center. For security, this system deploys a hierarchical identity-based cryptography and a role-based access control scheme. In the access control scheme, the root Key Generation Center (KGC), domain KGC, and sub-level domain KGC are proposed, tackling the inconvenience created when accessing services from different domains. With the layered KGC design, isolation of domain services is better achieved.

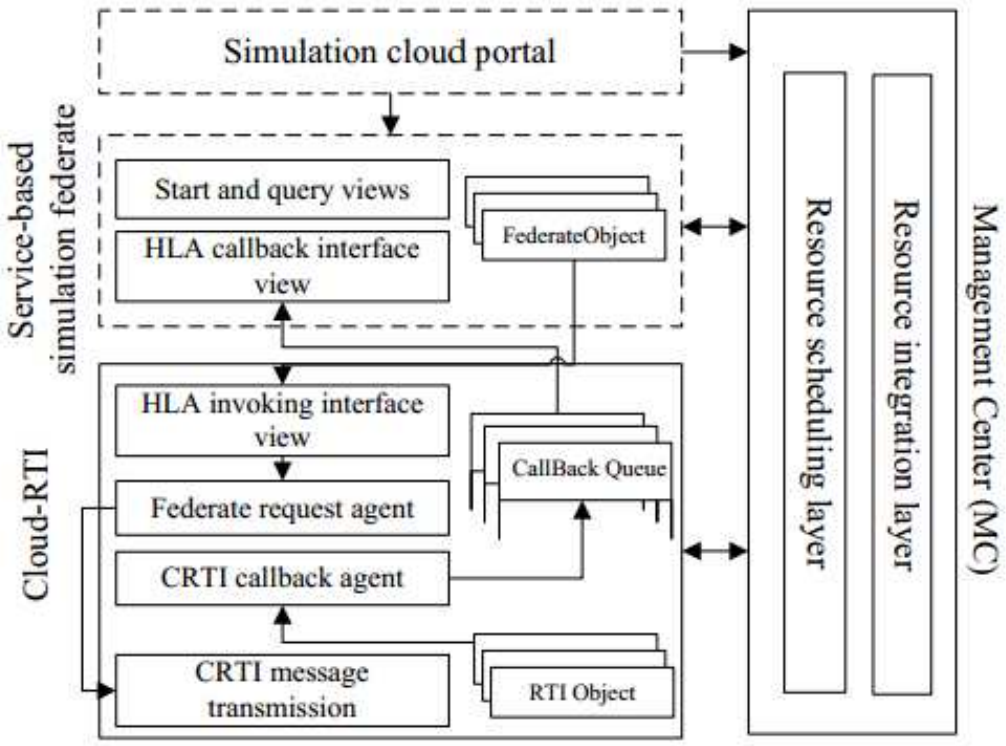

Figure 3.3: Cloud-RTI Framework

3.2.6 RACS

The Cloud Simulation System in [62] attempts to draw a full-scale picture of the cloud-based simulation system. In terms of security, it integrates portal security agents, access control and self-organization to enhance protection of sensitive data. The security control domain is defined with different levels of privilege, in attempts to avoid malicious behavior. A management system is created for various resources, recording global ID and life-cycle and attempting to ease resource allocation. Individuation virtual desktop technology, multi-user oriented dynamic building technology, and automatic composition

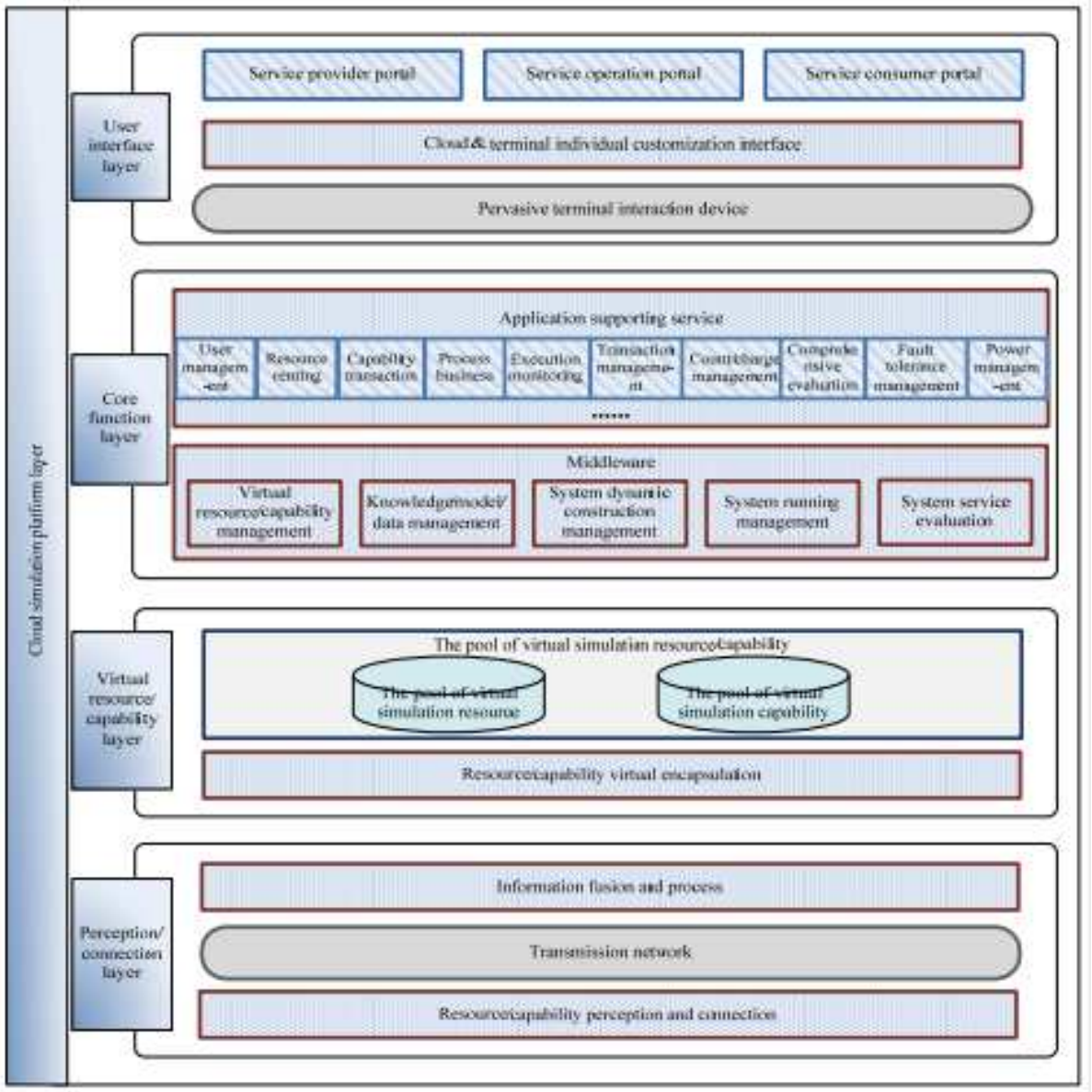

Figure 3.4: Cloud Simulation System Framework

technology of simulation models are used to support multi-user operation and simulation environment deployment. A parallel engine is defined to support fine-grained resources and tackle the sub-model issue in one federate. In this engine, lookahead is defined to extend the time interval of model parallelism, and tread-level simulation engine instances are created for large-scale simulations. Monitoring and evaluation technology is implemented to detect abnormal behaviors on the platform and provide information for further optimization decisions.

3.2.7 Summary

Jung [60] and He [56] discuss the performance of the application's execution. Comparisons are made between simulations of cloud virtual instances and simulations on native hosts. The results show that Cloud-based platforms achieve similar performance in terms of simulation execution. In [48] by Feng, time-latency is compared between Cloud-based RTI and native Portico RTI [72]. The author shows that, within a certain range of length of update data, time latency is acceptable on the cloud.

As shown in Table 3.1 and 3.2, the aforementioned cloud simulation schemes promote many aspects of distributed simulations in the cloud environment. However, these schemes offer limited details regarding the deployment procedure for the proposed cloud simulation platform itself, which is challenging for modelers who may not be familiar with the management of the distributed system. As a result, the cloud-based simulation is relatively hard to popularize due to the complex configuration and maintenance of di-

Table 3.1: Comparisons in Cloud Simulation Platforms 1

Cloud Simulation Platform	Job Scheduling	Security	Usability	Consumption
MapReduce	Yes			
Aurora System	Yes			
TW-SMIP	Yes			
CSim	Yes			
CDS		Yes		
Li, Bo Hu et al.		Yes	Yes	
Jung, In-Yong et al.				
Feng, Shaochong et al.				
Proposed Scheme	Yes	Yes	Yes	Yes

verse raw resources which constitute the fundamental computing capability of the cloud environment. In addition, the current cloud simulation platforms are mainly focused on the private cloud deployment model, which means only limited local physical raw resources are virtualized and coordinated in the cloud resource pool. In this case, for end users the compute utility seems inadequate, particularly for large-scale distributed simulations. Meanwhile, the energy consumption issue is not fully addressed and discussed for the distributed simulation in the cloud resource pool, which may result in a great deal of waste when the scale of computing resources are increased.

In this thesis, a multi-layered elastic cloud simulation scheme is proposed that addresses security, usability and job scheduling. The simulation performance is evaluated

Table 3.2: Comparisons in Cloud Simulation Platforms 2

Cloud Simulation Platform	Comparison Experiments	Portability	Scheme Design
MapReduce	Yes		
Aurora System	Yes		
TW-SMIP	Yes		
CSim			Yes
CDS	Yes		Yes
Li, Bo Hu et al.			Yes
Jung, In-Yong et al.	Yes		Yes
Feng, Shaochong et al.	Yes		
Proposed Scheme	Yes	Yes	Yes

and compared to the native Grid system. In addition, the concept of energy consumption and inter-cloud portability issues are fully concerned. A fast deployment model is proposed to enable quick installation of the platform. A migration-based algorithm is implemented to centralize simulation tasks and halt idle resources in order to conserve energy.

Chapter 4

The Cloud Simulation Platform

This multi-layered platform is proposed to support diverse distributed simulation standards and to hide the management of underlying details. A deployment model is designed to install, migrate and duplicate the platform in new environments easily. A web-based graphic portal with safety strategies allows users to get access to the platform workbench through light weight terminals. An self-managed simulation resource pool is implemented that supports, stores and configures simulation-related softlayer resources automatically for user's direct usage. An integration and virtualization approach is designed that can handle hardlayer resources as plug-in components elastically, requiring no direct configurations. An Energy saving mechanism is also utilized to reduce unnecessary energy cost.

As shown in Figure 4, this Cloud Simulation Platform consists of five layers: the Raw Resource Layer, the Integration and Virtualization Layer, the Simulation Function

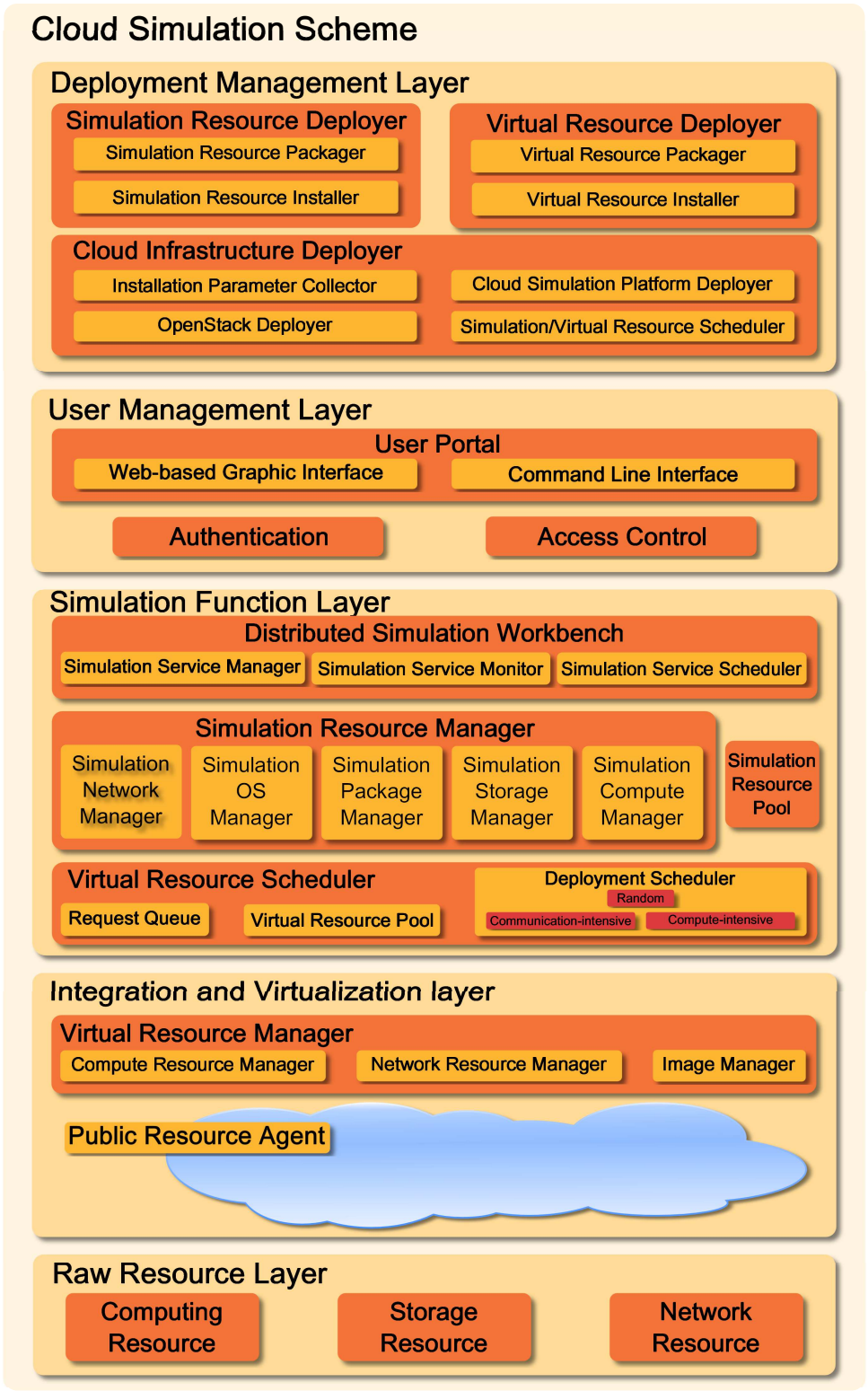

Figure 4.1: Cloud Simulation Scheme

Layer, the User Management Layer and the Deployment Management Layer. Specifically, deployment mechanism, security mechanism, migration mechanism and resource scheduling mechanism will be illustrated in detail with the description of related key features of each layer.

4.1 Deployment Management Layer

In order to provision fast and automatic deployment and migration of this proposed cloud scheme in new environments, the Simulation Resource Deployer, the Virtual Resource Deployer and the Cloud Infrastructure Deployer, as depicted in Deployment Management Layer of Figure 1, cooperate to back-up current platform status, pack and transmit core resources to new environments and deployment the platform based on previous saved status.

4.1.1 Simulation Resource Deployer

The Simulation Resource Deployer is focusing on simulation-related package handling. These simulation-related packages are managed by Simulation Storage Manager (in Layer 3, Figure 4) and are stored in both cloud and block storage. When platform packaging starts, the Simulation Resource Packager is triggered by the packaging process and it calls the Simulation Storage Manager to find all packages that are required to be packed. After the searching completes, the Simulation Resource Packager compresses all necessary packages, waiting for transmitting. When installation begins, the Simu-

lation Resource Installer is invoked by the installation main program. It decompresses simulation-related packages and stores these packages in one temporary folder. After the Cloud Infrastructure is properly deployed, the Simulation Resource Installer calls Simulation Resource Manager to register these packages currently stored in temporary folder. Finally, these packages are removed from temporary folder and stored in proper cloud storage and block storage again, ready for further usage.

4.1.2 Virtual Resource Deployer

The Virtual Resource Deployer functions similarly as the Simulation Resource Deployer. It is responsible for the handling of the simulation units – the virtual instances. When the packaging starts, it calls the Virtual Resource Scheduler (in Layer 3, Figure 4) and the Virtual Resource Manager (in Layer 4, Figure 4) to record real-time status of all computing instances in the whole system. The current status of virtual instances are saved by creating images. The Virtual Resource Packager classifies these images according to its owners. During the installation process, the Virtual Resource Installer first checks the Authentication status of the image owners, and then unpacks the images based on image users. Before the image can be used in the new environments, the Virtual Resource installer needs to trigger the Virtual Resource Scheduler to register the images and rebuild computing instances accordingly. These instances are created with the same instance type and network topology as they are in the original cloud simulation platform.

4.1.3 Cloud Infrastructure Deployer

The Cloud Infrastructure Deployer disposes components related to cloud middleware and virtualization tools. The configuration of these components are highly based on the new targeting environments. The Installation Parameter Collector is designed to collect relevant parameters required during the installation for the new physical environment, such as compute capability, Hard Disk capacities, network characteristics. These parameters are passed to the OpenStack Deployer and the Cloud Simulation Platform Deployer to install the core components and services as shown in Figure 4. After this step, the Simulation/Virtual Resource Scheduler triggers the Simulation Resource Installer and the Virtual Resource Installer to handle relevant softlayer and hardlayer deployment respectively.

4.1.4 Cloud Simulation Platform Packaging and Installation Algorithm

The whole packaging and installation process is shown in Algorithm 1 and 2. After the packaging main process is initialized, the Cloud Infrastructure Deployer calls the Simulation Resource Manager, the Virtual Resource Manager and the User Portal to stop accepting new tasks as shown in line 6 Algorithm 1. Once the current simulation tasks and the virtual instance scheduling tasks complete, two sub-processes are triggered by the main program to package simulation resources and virtual resources at the same time. On one hand, the Simulation Resource Packager finds the required simulation-related

Algorithm 1: Cloud Simulation Platform Packaging Algorithm

```
1  Require: SIGN_SR, SIGN_VR, SIGN_CP.
2  if SIGN_CP == packaging_start
3  then
4      SIGN_SR ⇐ SIGN_CP
5      SIGN_VR ⇐ SIGN_CP
6      CID.stop(UM, SRM, VRM, SIGN_CP)
7      while SIGN_CP! = packaging_ready
8      do
9          SRpacks ⇐ SRP.search(SRM, SIGN_SR)
10         SRpacks.save()
11         VRpacks ⇐ VRP.search(VRM, IM, SIGN_VR)
12         VRpacks.save()
13         if SIGN_SR == ready && SIGN_VR == ready then
14             CID.add(SRpacks, VRpacks)
15             CID.save()
16             SIGN_CP ⇐ packaging_ready
17         else
18             SIGN_CP ⇐ recheck
19             SIGN_SR ⇐ SIGN_CP
20             SIGN_VR ⇐ SIGN_CP
21     SIGN_CP ⇐ transmitting
```

packages in the Simulation Resource Pool with the help of the Simulation Resource Manager. These packages are then copied and compressed from the cloud storage. On the other hand, the virtual computing resources are handled by the Virtual Resource Packager. The Virtual Resource packager calls the Virtual Resource Manager to find the virtual instances, and invokes the Image Manager to build images to save the status of each instance. After both sub-programs complete as shown in line 13, the packed simulation resources and virtual resources are registered to the main packaging process in the Cloud Infrastructure Deployer. Finally, the Cloud Infrastructure Deployer packs external tools suck as the cloud middleware and virtualization components, ready to

Algorithm 2: Cloud Simulation Platform Installation Algorithm

```
1  Require: SIGN_SR, SIGN_VR, SIGN_CP.
2  if SIGN_CP == installation_start
3  then
4      CID.auth()
5      IP ⇐ IPC.search(comp, st, nw)
6      OD.install(IP)
7      CSPD.install(IP)
8      SVRS.sche(SRI, VRI)
9      while SIGN_CP! = installation_end
10     do
11         SRI.unpack(SRpacks)
12         SRI.register(SRpacks)
13         VRI.unpack(VRpacks)
14         VRI.register(VRpacks)
15         if SIGN_SR == installation_end
16         && SIGN_VR == installation_end
17         then
18             CID.test()
19             CID.end()
20         else
21             SIGN_CP ⇐ recheck
22             SIGN_SR ⇐ SIGN_CP
23             SIGN_VR ⇐ SIGN_CP
```

transmit.

The installation process contains three steps, as shown in Algorithm 2, the cloud infrastructure installation, the simulation resource installation, and the virtual resource installation. The main installation process first invokes the Cloud Infrastructure Deployer to build the basic cloud platform. Before the real installation process begins, the installation parameters that are required during cloud infrastructure installation are collected by the Installation Parameter Collector. The collecting process may need the authorization of the system administrator before it can start. Then, the OpenStack Deployer

and the Cloud Simulation Platform Deployer build core components in the Integration and Virtualization Layer and main services in the Simulation Function Layer. After this, The Simulation/Virtualization Resource Scheduler calls the Simulation Resource Installer and the Virtual Resource Installer to install simulation resources and virtual resources. The Simulation Resource Installer and the Virtual Resource Installer work simultaneously, unzipping, storing and registering relevant packages in the new environments. After these two sub-processes are finished, in the end, the Cloud Infrastructure Deployer tests the functions of key services, and the availability of simulation pool and virtual resource pool, ensuring the correct installation.

4.2 User Management Layer

This layer is focusing on the security issues in the cloud environment. In this layer, an authentication and access control mechanism is proposed to protect users against threats from both inside the cloud and outside the cloud. In addition, this layer also provides a web-based graphic portal and a command-line interface, through which users can get access to the simulation resources and the computing capability of the proposed cloud simulation platform. This user portal enables users to design, code, analyse and test complex distributed simulations via light weight terminals such as laptops, tablets or even smart phones.

4.2.1 Authentication and Access Control Algorithm

The Algorithm 3 shows the authentication and access control process. After the user registers in the platform, as shown in line 2, the cloud firewall adds the new user in the acceptance list. In this stage, the user is authorized the right to access to the shared materials in the Simulation Resource Pool. Users can take advantage of softlayer resources in the pool that gathers and presents codes, templates, solutions, applications and experiences shared by other cloud users. In order to secure the user-defined simulations in the cloud simulation platform, one unique private key is created and stored by the simulation owner, which is used to further encrypt user's personal operations and other sensitive data. An X-token controlled by the Virtual Resource Manager is also provided to the user that enhances the security of virtual instances. It is an access key for the user to access its personal virtual instances in the Virtual Resource Pool, as shown in line 11. It is required when users create, access to, modify or delete instances. With encryption, X-token and virtualization technology, users on the cloud are better isolated and protected so that malicious operations from cloud users or cloud administrators inside the cloud are able to be better blocked, ensuring that at any time, one virtual instance can only belong to one cloud user for its exclusive usage. This X-token expires within 24 hours by default, and one new token can be required if necessary. In this case, if one user is accidentally attacked and loses the control of its instances, a x-token reset command is able to stop the hacker from tapping into these personal data for further destruction.

Algorithm 3: Authentication & Access Control Algorithm

```
1  Require: username/password, private_key, x_token
2  User.signup()
3  User.fw.ad(user)
4  if User.auth() == True
5  then
6      User.access.add(SRP.r)
7      if User.auth(key)
8      then
9          User.access.add(SRP.w)
10         if User.auth(x_token)is not expired then
11             User.access.add(VRP.w)
12         else
13             User.auth.update()
14 else
15     User.access.del()
```

4.3 Simulation Function Layer

This layer provides core functions and services that naturally enable and support different types of distributed simulation standards, such as HLA (High Level Architecture) [80], DIS (Distributed Interactive Simulation) [2] and DDS (Data Distributed Service) [67], which include the Distributed Simulation Workbench, the Simulation Resource Manager, and the Virtual Resource Manager.

4.3.1 Distributed Simulation Workbench

The Distributed Simulation Workbench offers to users a self-defined distributed simulation foreground, enabling users to focus on designing, analysis and testing without further concerns regarding the configuration and maintenance of underlying physical system.

Based on the user's privilege, the Simulation Service Manager furnishes and coordinates simulation services for users, which include online modelling service, simulation analysis service, fault tolerance service and load relocation service. Online modelling service is designed for modellers to code, submit, execute and debug through a web-based light weight interface. Other services are able to be lightly invoked by this interface. Simulation analysis service records and provides to end users the results of simulation execution, simulation-related system logs, and simulation performance statistics such as CPU load, simulation execution time, and network bandwidth usage, helping to reveal potential issues in the user-designed programs and applications. Fault tolerance service protects user-defined simulations from computation errors, storage errors and network errors, providing roll back and fail over mechanisms. The Virtual Machine (VM) based migration is utilized as an approach for simulations to recover from failed physical resources. Load relocation service supports the ability for users to reschedule virtual instances, modify instance types, or network topology during run-time, simplifying the set of comparison experiments. The status of these services are tracked by the Simulation Service Monitor. In addition, the Simulation Service Monitor detects and updates the states of the physical machines that compose these simulation services periodically. If any physical error occurs, the Simulation Service Monitor is responsible to block relevant resources and inform the cloud simulation platform administrator. The Simulation Service Scheduler manages and maintains the background execution queue for the simulation service requests from multi-users, ensuring proper execution orders of simulations. In addition, the

Simulation Service Scheduler calculates the whole platform system energy consumption and utilizes energy efficiency policies to reduce energy cost by migrating and centralizing virtual instances and releasing idle physical machines.

4.3.2 Simulation Resource Manager

The Simulation Resource Manager concerns the network, OS, software dependency, storage and computation aspects of the distributed simulations. It also manages the Simulation Resource Pool, which gathers, presents and shares codes, templates, solutions and applications of diverse distributed simulation standards among the cloud users. The Simulation network manager handles the network model for the given distributed simulations. It currently supports unicast, multicast and broadcast approaches that most distributed simulation standards utilize for communication. In addition, it works with the Virtual Resource Scheduler to initialize virtual instance's network topology for self-defined virtual instances. The simulation OS Manager controls and stores a list of images of OS for booting virtual instances. Users are able to modify system settings such as firewall rules, partitioning of hard disk, software repository and other system information before the virtual instances are booted. The Simulation Package Manager is responsible for arranging simulation depended packages. It natively supports standards including HLA, DIS, DDS, and automatically configures and installs dependency packages that each corresponding implementation may further require. In order to enable the diversity of distributed simulations, the Simulation Package Manager also allows users to manage

and define self-designed packages for specific scenarios. The Simulation Storage Manager provides two types of storage in the background for simulations: the cloud storage and the block storage. The cloud storage is mainly used as replicas to backup resources in the Simulation Resource Pool in case that local resource storage servers are down. In addition, these cloud replicas are able to be easily shared among users inside and outside the platform. The block storage saves all necessary information that this cloud simulation platform requires during run-time and it utilizes the Network File System (NFS) to accelerate the scheduling process of simulation resources. The Simulation Compute Manager concerns the compute capability for the current distributed simulation. It communicates with the Simulation Service Monitor to check local compute resources. If local resources are not sufficient for current simulations, it calls the Virtual Resource Manager and the Public Resource Agent to enable additional public computing resources to fulfil the computation requirements.

4.3.3 Virtual Resource Manager

Due to the diversity of modelling and simulation applications, requirements of computing resources differ. The Virtual Resource Scheduler is utilized to control the granularity of resources so that the need of diverse simulation applications can be better met. Based on the requirement information from users, the Virtual Resource Scheduler helps to arrange the simulation environment based on parameters of the computing unit such as the number of virtual processors, memory size, network topology, and the quantity of virtual

machines. The Virtual Resource Scheduler invokes corresponding components in the Virtual Resource Manager and in the Public Resource Agent, discovering proper computing resources and instantiating virtual computing instances in the Virtual Resource Pool.

4.3.4 Scheduling Algorithm and Dynamic VM Reallocation Algorithm

Several resource scheduling algorithms are provided for instance creation, as shown in Algorithm 4 : random, computation-intensive, and communication-intensive. N in line 1 defines the number of virtual resources that require to be deployed for simulations. The default value is 0, which means the current available resources are already adequate and no additional virtual resources are needed to be scheduled from the Virtual Resource Pool. In this context, IT in line 1 is short for the instance type. This parameter expresses the type of virtual instances that may be further initiated and utilized. It contains computing and storage capability information such as processors, RAM, hard disk. SA, which is first shown in line 1, stands for the type of the scheduling algorithm. In Random, as shown from line 4 to line 9, the Virtual Resource Scheduler creates instances randomly. First, from the Virtual Resource Pool, a queue of all current available resources are listed. Then, based on the number of available resources, a random integer number n is selected. The resource whose index number is n in the resource queue is chosen as the destination physical host. After this selection, the request of booting the virtual instance is queued in background and then redirected to the destination physical host. Finally,

the virtual instance is instantiated on the targeting physical host, and the system records and updates the resource information in the Virtual Resource Pool. At this time, one round of scheduling ends and the next round is ready to begin. In computation-intensive scheduling, shown from line 10 to line 15, a current available resource queue is first created in the same manner as random. Then, the resource whose current computing capability is the largest in the queue is selected as the destination resource. In this mode, typically large virtual instance types are chosen as IT for simulations involving large computation tasks. With this type of scheduling and large virtual instance type, the rounds of scheduling can be tremendously reduced. In communication-intensive mode shown between line 16 and line 22, the first resource in the current available resource queue is selected as the destination host. Then, this selected physical host attempts to schedule and boot as many virtual instances as possible on itself until its compute capability is exhausted. In this way, communication-intensive simulation entities can be more likely to exist in the same physical host so that the communication distance of simulation entities is reduced. After each round of scheduling, a virtual instance migration may be triggered by the Simulation Service Scheduler, reallocating virtual instances and centralizing simulation tasks. In this case, idle physical resources can be released and the energy consumption of the system can be reduced.

Algorithm 4: Scheduling Algorithm & Dynamic VM Reallocation

```
1  Require: N, IT, SA
2  N_update(), IT_update()
3  if SA is Random Scueduling then
4      while N! = 0 do
5          ava_res_que ⇐ res_pool.search(IT)
6          n ⇐ random(0, ava_res_que.index)
7          des_res ⇐ ava_res_que.get(index = n)
8          boot_que.add(des_res, IT)
9          sys.rec(), sys.update(), N_update(), VM.realoc()
10 if SA is Compute − intensive Scueduling then
11     while N! = 0 do
12         ava_res_que ⇐ res_pool.sort(IT)
13         des_res ⇐ ava_res_que.find_low_load()
14         boot_que.add(des_res, IT)
15         sys.rec(), sys.update(), N_update(), VM.realoc(),
16 if SA is Communicate − intensive Scueduling then
17     while N! = 0 do
18         ava_res_que ⇐ res_pool.search(IT)
19         des_res ⇐ ava_res_que.get(index = 1)
20         while des_res is not exhausted or N! = 0 do
21             boot_que.add(des_res, IT)
22         sys.rec(), sys.update(), N_update(), VM.realoc(),
```

4.4 Ingegration and Virtualization Layer

In order to utilize the raw compute capacity from diverse resources, the Integration and Virtualization Layer is designed to coordinate these resources and implement virtualization technologies. This layer contains two core components: the Virtual Resource Manager and the Public Resource Agent. Although these components are not visible to end users, they play an important role in automatic configuration, management and maintenance of the underlying raw resources, hiding complex details of system management.

4.4.1 Virtual Resource Manager

The Virtual Resource Manager plays a crucial role in integrating physical resources, which includes three sub-managers: the Compute Resource Manager, the Image Manager, and the Network Manager. It maintains and manages the cloud middleware and virtualization tools. Based on the user-defined requirements from the aforementioned Virtual Resource Scheduler, the Virtual Resource Manager calls its relevant sub-managers to process compute-related, storage-related or network-related management program, building the experiment environment. The Compute Resource Manager deals with the establishing procedure of virtual instances based on the selected scheduling algorithm and the characteristics of the user-defined virtual instances. The Image Manager handles the OS and simulation-related soft-layer issues. According to different implementations of different distributed simulation standards, the Image Manager pre-arranges the operation system and the simulation-related packages for the virtual instances before it is booted from the Compute Resource Manager. The Network Manager controls the network topology of the virtual instances. In this case, these virtual instances are able to be instantiated in the same physical host or among multiple adapters, based on the simulation requirements. In addition, the Network Manager maintains the key bridge that is designed to support communications between local virtual instances and public instances. It virtualizes the public portal of the cloud simulation platform, binding the virtual portal and public tunnels.

4.4.2 Public Resource Agent

In order to achieve rapid elasticity in the cloud simulation platform, the Public Resource Agent is designed as the component to increase computing capability during run-time by introducing public physical resources such as instances in the public cloud – Amazon EC2 instances. To coordinate the local resources and public resources, the diversity of public network should be taken into consideration carefully. In the public network, multicast and other approaches of package transmission protocols, which are widely used in many distributed simulation standards, may not be well supported by routers, switches or hubs. For example, routers drop multicast messages by default, if the destination resource and the original resource are not set in the same subnet. The Public Resource Agent is introduced to manage the public networks to meet the demand of distributed simulation. First, the Public Resource Agent builds specific virtual tunnels for each public resource based on the physical network, penetrating the public Internet and encapsulating packages and messages that need to be sent and received. After the virtual tunnels are initiated, the relevant Firewall rules are modified to support the established communication among local instances and public instances, redirecting packages and messages to proper destinations. The Public Resource Agent listens to the Virtual Resource Scheduler and analyses user's simulation requirements, only invoking public instances if local computing capability is not sufficient and user's Service Level Agreement is not violated.

4.4.3 Simulation Resource Provisioning and Mapping Algorithm

The simulation environment preparation contains mainly two sub-processes, as shown in Algorithm 5 and 6. The underlying simulation execution process is revealed step by step. First, as shown in Algorithm 5, when the Virtual Resource Scheduler receives simulation tasks from the user, the user's identity and priority is verified, shown in line 3, ensuring user's full access to the Simulation Resource Pool and the Virtual Resource Pool. The detailed authentication and access control process is demonstrated in Algorithm 3. Then, based on the collected information from the Simulation Service Monitor, the Virtual Resource Scheduler decides whether it is necessary to invoke public instances. On one hand, If local computing capabilities are sufficient, shown in lines 5 to 10, the Public Resource Agent is not contacted. In this situation, the Virtual Resource Manager calculates the number of virtual instances that require to be instantiated and collects user-defined parameters such as SA, IT, and simulation related packages. Then the Virtual Resource Manager boots the instances in line 9, as illustrated in Algorithm 5. On the other hand, if local computing resources is not enough for the requirement, the Public Resource Agent is called to arrange additional computing capabilities, shown in lines 12 to 17. In this case, after the local computing resources are exhausted, the Public Resource Agent configures, links and bridges the public instances to local user groups, enabling communications among local virtual instances and public instances in the same user-defined group. In this stage, the virtual resources are properly prepared, ready to execute softlayer commands.

Algorithm 5: Simulation Resource Provisioning

```
1  Require: VRP.w, N, IT, SA, P.
2  Note: VRP − Virtual Resource Pool, P − Simulation related Packages,
   VRS − Virtual Resource Scheduler, VRM − Virtual Resource Manager,
   SRM − Simulation Resource Manager, CRM − Compute Resource Manager,
   IM − Image Manager, NM − Network Manager, PRA − Public Resource Agent,
   SRP − Simulation Resource Pool.
3  while !VRS.end() && User.access(VRP.w) do
4      while VRS.cur(VRP) <= VRS.req() do
5          if VRS.req() <= VRS.cur(VRP.loc())
6          then
7              VRM.N.set() ⇐ VRS.req() − VRS.cur(VRP))
8              VRM.IM.set(P) ⇐ SRM.map(SRP)
9              VRM.boot(VRM.CRM(N, IT), VRM.NM(SA), VRM.IM(P))
10             VRS.update()
11         else
12             VRM.N.set() ⇐ VRS.req() − VRS.cur(VRP.loc())
13             VRM.IM.set(P) ⇐ SRM.map(SRP)
14             VRM.boot(VRM.CRM(VRS.cur(VRP.loc()), IT),
               VRM.NM(SA), VRM.IM(P))
15             PRA.set(VRM.N, PRA.getpair(PL), VRM.IM(P))
16             VRM.NM.bridge.setgroup(VRM.cur(user), PRA.cur(user))
17             VRS.update()
18     VRS.end() ⇐ User.def()
```

Before the distributed simulation applications begin to run, the simulation scripts are supposed to map to destination instances by the Simulation Resource Manager, shown in Algorithm 6. The Simulation Resource Manager pairs the index of user-defined simulation packages in the Simulation Resource Pool and the index of virtual instances in the user's private Virtual Resource Pool, delegating simulation executing packages to destination instances. After each round of simulation, the executing packages can be remapped to different instances for comparison study. In the end, after the simulations completes, shown in lines 11 to 15, the simulation-related packages and logs are uploaded

to Hard Disk storage and cloud storage. The status of computing resources is saved as images and the computing capabilities are recycled.

Algorithm 6: Simulation Resource Mapping

```
1  Require: VRP.w, N, IT, SA, P.
2  Note: VRP − Virtual Resource Pool, P − Simulation related Packages,
   VRS − Virtual Resource Scheduler, VRM − Virtual Resource Manager,
   SRM − Simulation Resource Manager, CRM − Compute Resource Manager,
   IM − Image Manager, NM − Network Manager, PRA − Public Resource Agent,
   SRP − Simulation Resource Pool.
3  while !SRM(end)&&User.access(VRP.w) do
4      VRM.insList.set(user) ⇐ VRM.cur(VRP.user.simX)
5      SRM.simList.set(user) ⇐ SRM.upload(SRP.user.simX)
6      SRM.mapSimPair(VRM.insList(user), SRM.simList(user))
7      SRM.execute()
8      SRM(end) ⇐ User.def()
9  if SRM(end)
10 then
11     SRM.upload()
12     SRM.SRP.save()
13     VRM.IM.save()
14     VRM.recycle()
15     User.access ⇐ NONE
```

4.5 Raw Resource Layer

The Raw Resource Layer is a pool of untreated resources such as computing resources, storage resources, and network resources. The whole cloud simulation platform is build on these resources. In this layer, physical hosts are not configured and coordinated, only containing an OS and some basic software that such OS brings.

Chapter 5

Experiment Results and Discussion

Based on the aforementioned architecture, a serious of experiments are implemented on the cloud simulation platform in the cluster, single machine and public cloud instances respectively, concerning the simulation environment preparation, job scheduling, energy consumption and execution efficiency.

5.1 Experiment Environments

The prototype of this platform has been implemented and deployed on a cluster, on one single machine, and on the Amazon EC2 instance respectively to test and compare the performance.

The cluster was composed of 20 nodes, and each node of the cluster contained a QuadiCore 2.40GHz Intel(R) Xeon(R) CPU and 8 gigabytes of DIMM DDR RAM memory. All nodes were inter-connected through a Myrinet optical network that allowed data

transmission up to 2 gigabytes per second.

The single machine setup contained a QuadiCore 2.40GHz Intel CPU (4700MQ) and 16 gigabytes of RAM. VMware Workstation [84] was used as the virtualization tool which created one management node and two computing nodes. The details of the nodes are shown in Table 5.1.

For public cloud instances in our experiments, T2.micro instance from AWS [5] was employed, which provided a basic configuration: one VCPU and one GiB of memory.

In the experiments, the OpenStack [71] is used as cloud middleware and the KVM (Kernel-based Virtual Machine) [61] is used as the virtualization tool. Fedora and CentOS Linux systems are used as OS in the virtual instances.

Based on different deployment environments, instance types, scheduling algorithms, experiments were implemented to evaluate the energy consumption, resource provisioning efficiency, and simulation performance.

Table 5.1: Single Machine Environment

Host	**VCPU**	**RAM**	**Bandwidth**	**Hard Disk**
Management	2	2 G	100 Mbps	30 G
Computing 1	4	4 G	100 Mbps	40 G
Computing 2	1	2 G	100 Mbps	20 G

5.2 Performance Analysis of Environment Preparation

The first experiment concerned the preparation time of simulation environment that includes the computing resource scheduling time, raw resource providing time, and softlayer resource packaging time. The computing resource scheduling time involves the components of the whole platform: the scheduling time starts when the User Portal receives the user's resource requests and ends after these requests are properly stored, recorded, and processed in the Request Queue managed by the Virtual Resource Manager. The rare resource provision process includes the selection of computing resources in the raw computing pool, the mapping of OS and computing resources, and the establishing of instances. The softlayer resource packaging procedure focuses on the building of a simulation-run-ready environment, which contains the simulation-related package dependency analysis, package searching, package configuration, and package installation. In this experiment, Linux systems was used as the operating system for the virtual instances; CentOS 6 [39] and Fedora 17 [47] were used. The simulation applications used one testing restaurant simulation application implemented in Portico java-based RTI [72] as the HLA simulation; one simple Sender/Receiver application implemented in OpenDIS [1] as the DIS simulation; and one "HelloWorld" application of OpenSplice [74] as the DDS simulation. The virtual instance type for the cluster and the single machine employed in this experiment is shown in Table 5.2. In this experiment, a random scheduling

was selected as the scheduling algorithm for the computing instances.

Table 5.2: Virtual Resource List

Virtual Instance Type	VCPUs	RAM(MB)	Hard Disk(GB)
Mini	1	512	0
Small	1	1024	10G
Medium	2	2048	20G
Large	4	4096	40G
Self-defined	1 to 4	512 to 4096	0 to 40G

The results of experiment one are shown in Table 5.3, 5.4, and 5.5. The $Time\_sche$ refers to the computing resource scheduling time, the $Time\_f17$ and $Time\_cen6$ means the raw resource providing time with a Fedora 17 OS and a CentOS 6 OS respectively in Table 5.3 and 5.4. As the result shows, on one hand, the cluster mode achieves a better performance in resource scheduling due to the better computing capability of the management node. On the other hand, the raw resource providing time of the single machine mode is much shorter because its scale of computing resource pool is small, which only contains two computing nodes. In this case, the raw resources can be recycled, regenerated, and reused more efficiently after they are released for each round. The results in Table 5.5 shows the softlayer resource packaging time. It indicates that based on this proposed cloud simulation scheme, a simulation-run-ready environment can be completely brought up and ready for executing within 250 seconds. Different types of distributed simulation standards, operating systems, and computing instances are auto-

matically configured, instantiated and managed based on the simulation requirements.

Table 5.3: Computing Resource Preparation in the Single Machine Mode

Virtual Resource Type	Time_sche(s)	Time_f17(s)	Time_cen6(s)
Mini	2.064	3.445	3.945
Small	2.278	3.705	3.998
Medium	2.162	3.753	4.011

Table 5.4: Computing Resource Preparation in the Cluster Mode

Virtual Resource Type	Time_sche(s)	Time_f17(s)	Time_cen6(s)
Mini	1.389	16.871	17.539
Small	1.421	17.278	17.932
Medium	1.322	17.502	18.244
Large	1.433	17.892	18.577

5.3 Performance Analysis of Job Scheduling

The second experiment evaluated the performance of scheduling algorithms that were natively supported in this cloud scheme when simulations scaled up in the cluster mode. In this experiment, mini virtual instances and large virtual instances were introduced as computing instance examples. The scheduling time recorded in this experiment was the same as T_sche, explained in the first experiment.

Table 5.5: Packages Handling Process

Platform Mode	Time_hla(s)	Time_dis(s)	Time_dds(s)
Single Node	189.845	141.728	234.013
Cluster	144.672	97.412	189.844

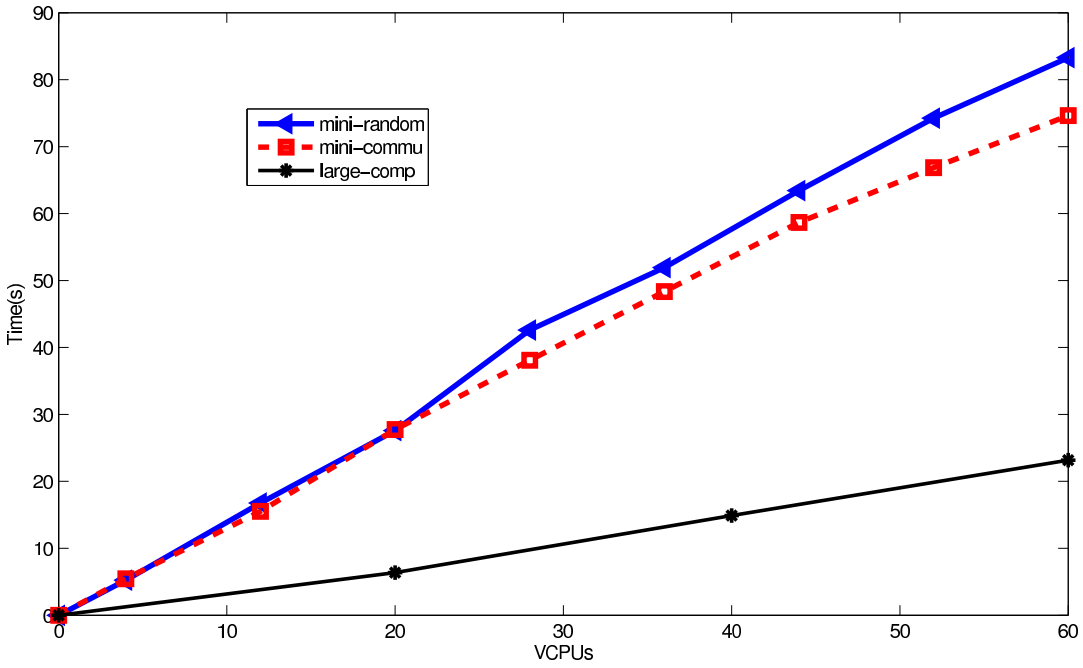

Figure 5.1: Scheduling Algorithm Performance Evaluation

As depicted in Figure 5.1, with simulations scaling up, the time of scheduling increased almost linearly, except for some slight fluctuations caused by background load. The background load was originated from the cloud system, which needed to keep running its services for periodically detecting and updating the status of the physical machines that compose the environment. Such services coordinated the resources by observing and recording their power status, network availability, current free RAM, and other character-

istics. As the result shows, Computation-intensive scheduling presented the best performance because it minimized the rounds of resource scheduling. As to random scheduling and communication-intensive scheduling, the latter presented an overall performance 10 percent better. This difference was generated because in the communication-intensive scheduling, the search queue for available resources in resource queue was shorter – the first available physical host in the resource queue was selected as the destination host. For random scheduling, the average scheduling search length was $que.len/2$, which was much longer than the communication-intensive. Another reason for obtaining these results might be the influence of a built-in optimization, which could have been possibly used for cutting resource recording time for communication-intensive scheduling. Communication-intensive scheduling recorded only when one physical host was exhausted while random scheduling needed to record right after each scheduling round finished. For different types of scheduling algorithms in this platform, the scheduling time for one instance was less than 1.5 seconds. Instead of configuring relevant simulation aspects manually, these automatic scheduling components of this platform can save modellers plenty of time.

5.4 Analysis on Energy Consumption

The third experiment regards the energy consumption of the proposed scheme. Due to the lack of temperature sensors in the hardware, CloudSim [38] was employed to estimate and observe the performance of the energy saving approach in the scheme. The

parameters in the experiment, such as the VM migration time, VM scheduling time, were based on the real test results of the machines in the cluster, shown in Table 5.6. The simulation used a long-term computing extensive example that consumed almost 100% CPU and run for approximately 24 hours. The types of computing instances are listed in Table 5.2.

Table 5.6: CloudSim Simulation Parameters

Scale	A cluster of 20 computing nodes and 1 management node
Number of VM Required	10 (Small, Medium, and Large)
VM Migration Time	23 Seconds
VM Scheduling Time	1.4 Seconds
Rated Power	355 W
Scheduling Time Interval	60 Minutes
Scheduling Algorithm	Random Scheduling

The Tables 5.7 and 5.8 show the results of simulation task scheduling and executing process with and without energy-aware components. As the results indicate, the additional migration and energy saving scheme can reduce almost half of the energy consumption by centralizing simulation tasks on physical hosts and releasing idle computing resources. In the experiment scenario, only approximately 300 seconds were consumed by the migration for all the sample simulation applications for a period of 10 hours, which was acceptable for these distributed simulations that run for periods of time in the scale of hours.

Table 5.7: Scheduling with Migration

Round	1 and 2	3 and 4	5 and 6	7 and 8	9 and 10
Current VM Type	S, M	S, S	S, M	L, S	L, M
Migration	1	1	2	0	0
Current Power	712.538	1058.662	1429.614	2485.944	3551.349
Current Hosts in Use	1	2	2	4	4

Table 5.8: Scheduling without Migration

Round	1 and 2	3 and 4	5 and 6	7 and 8	9 and 10
Current VM Type	S, M	S, S	S, M	L, S	L, M
Current Power	1065.54	2487.13	3909.884	5333.634	6758.49
Current Hosts in Use	2	4	6	8	4

5.5 Analysis on Simulation Performance

The fourth experiment concerned the performance loss in simulation execution that this multi-layered cloud platform may cause when compared with the native Grid platform. In the experiment, comparisons are made in the cluster mode, in the single machine mode, and in the public cloud computing instances from AWS [5]. In the cluster mode, the large virtual resource type as shown in Table 5.2, was used to boot fifteen virtual instances on the cloud while the grid particularly used fifteen physical hosts directly. In the single machine mode, a small computing instance type, shown in Table 5.2, was used for the cloud while the grid used the Host Computing 2 as shown in Table 5.1. The simulation

application in the experiment was implemented by the Portico java-based RTI [72]. The sample federate produced a controlled synthetic load which consists in first creating a pseudo-random number x and then introducing a recursive procedure for x turns to exhaust the compute capability on the instance where it ran. After the computation, the sample federates communicated with each other about their computation results. Different iterations were used to measure the simulation execution performance.

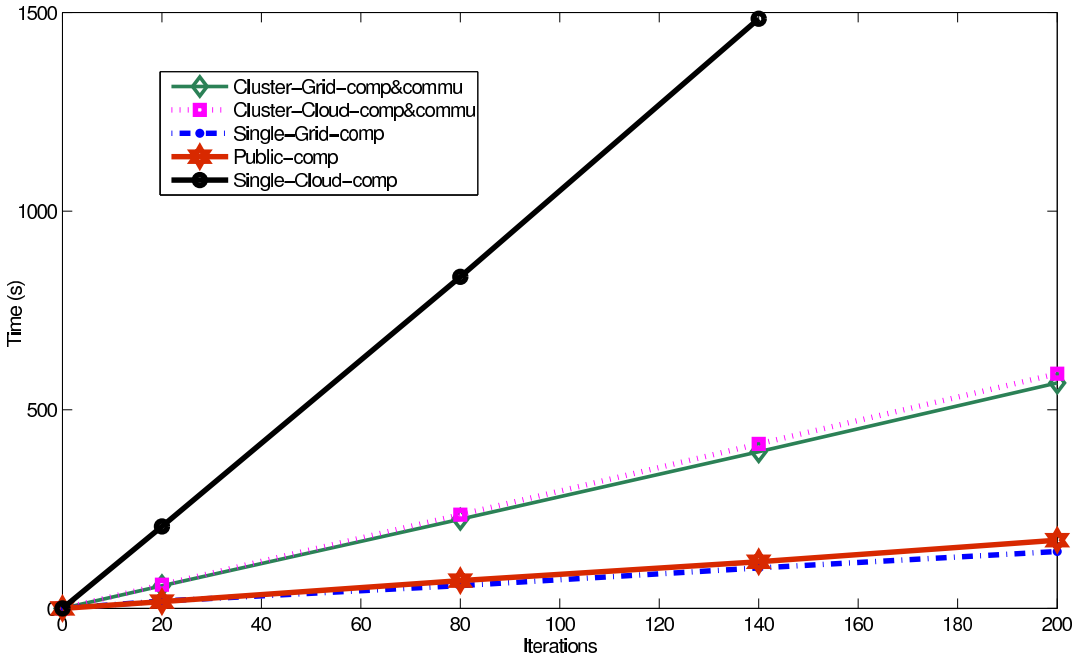

Figure 5.2: Simulation Performance between Cloud Platform and Grid Platform

The results for this experiment are depicted in Figure 5.2. In terms of the experiments in the cluster mode, which involved both communication tasks and computing tasks, the Grid outperforms the Cloud: when the experiment reached 200 iteration rounds, the performance of the Grid was 3.94% better than the Cloud. The reason for the

performance loss of the Cloud is the utilization of Virtualization Technology (VT) in the Integration and Virtualization Layer, according to [9]. As to the experiments in the single machine mode that only contained computing tasks, the performance of the Grid was much better: when the iteration number reached 140, the Grid performed approximately 15 times better than the Cloud. This results was obtained due to the nested virtualization for the cloud scheme in the single machine mode, which means creating an additional layer of abstraction and encapsulation in a pre-existing virtualized environment, occurred. This may influence the performance of executing simulations, according to the discussion and results from [10]. It is worth mentioning that for the public instance mode that contained the same computing task as the Grid in the single machine mode, the performance of the Amazon public cloud instance was only 19.8% slower than the latter when the iteration round reached 200, which indicates that the public cloud is very promising to hold computation-intensive distributed simulations.

5.6 Summary

Based on the results of aforementioned four experiments, we can observe that the proposed multi-layered cloud simulation platform supports elasticity, automatic management of diverse resources, fast deployment, security, and reduction of energy cost. There is little performance loss of simulation execution compared with the native platform mainly because of the virtualization of the Cloud. However. This overhead is acceptable due to the benefits of Cloud Computing listed in the early sections of this thesis.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, aside from the simulations of the cloud using cloud simulation tools such as [38], [55], [82] and [81], a real layered platform was proposed and implemented. The focus is on usability, energy consumption, security, reliability and elasticity. Relevant components are implemented to support fast deployment, ease the management of underlying resources, reduce energy consumption, and enable fine-grained resource handling during runtime. The design of a multi-layered scheme in different scenarios was described. In the experiments, the performance of this cloud simulation platform was evaluated and discussed in detail. Based on the experimental results, it was suggested that the use of cloud technologies is promising in facilitating distributed simulations, particularly when the network environment presents greater efforts towards optimization and performance.

6.2 Future Work

In future work, we will combine the live migration and fast deployment, attempting to solve the fault tolerance issue on the management node in the cloudcloud. Due to its heavy load in performing task scheduling, monitoring and partial communication, the management node is more vulnerable in real distributed simulations than computing nodes. A fast management node replication scheme will be implemented and evaluated, providing failover for both computing nodes and communication nodes. Moreover, in this proposed platform, the performance of public instances in Amazon Web Service have been presented and discussed. Due to the lack of standard API in Cloud Computing, integrating resources from different cloud service providers is challenging, particularly for communication-intensive applications that require frequent small message transmissions. The cross-platform issue in the cloud is addressed in [74], [71], and [65]. Based on the principles mentioned above, in the future, we will concentrate on the interoperability and handling high communication loads in the public cloud environment for the proposed cloud simulation scheme. The performance of the scheme will be evaluated by fully enabling distributed simulations in cross-platform public instances.

In the first stage, the cloud simulation platform supports main distributed simulation standards. In the future, more efforts will be directed towards the support for simulations on the wireless, ad hoc, and mobile networks, focusing on different simulators that are implemented regarding localization [44], [27], [4], [22], [23] [66], [28], [29]; congestion control [19]; performance efficiency [17], [25], [8], [77], [12], [13], [14]; detection [33], [26];

synchronization [24]; fault tolerance [16], [46]; security [21], [31], [75]; message routing [30]; and result evaluation [20], [15].

Bibliography

[1] *OpenDIS.* Available: http://open-dis.sourceforge.net/. Accessed on January 6, 2015.

[2] *IEEE Standard for Distributed Interactive Simulation–Application Protocols.* IEEE Computer Society. 1278-2012.

[3] *Globus.* University of Chicago. 7 Feb. 2008.

[4] O Abumansoor and A Boukerche. *A secure cooperative approach for nonline-of-sight location verification in VANET.* Journal of IEEE Transactions on Vehicular Technology, 61 (1). Page: 275-285, 2012.

[5] Amazon. *Amazon EC2 Instances.* Available: http://aws.amazon.com/ec2/instance-types/. Accessed on January 6, 2015.

[6] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. *A view of cloud computing.* Journal of Communications of the ACM. 53(4): 50-58. 2010.

[7] Denise Rotondi Azevedo, Ana Maria Ambrsio, and Marco Vieira. *HLA Middleware Robustness and Scalability Evaluation in the context of Satellite Simulators.* In proceedings of the 19th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC. Page: 312–317. 2013.

[8] A Bamis, A Boukerche, I Chatzigiannakis, and S Nikoletseas. *A mobility aware protocol synthesis for efficient routing in ad hoc mobile networks.* Journal of Computer Networks 52 (1). Page: 130-154, 2008.

[9] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. *Xen and the art of virtualization.* Journal of ACM SIGOPS Operating Systems Review, 37(5): 164-177, 2003.

[10] Muli Ben-Yehuda, Michael D Day, Zvi Dubitzky, Michael Factor, Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman, and Ben-Ami Yassour. *The Turtles Project: Design and Implementation of Nested Virtualization.* In proceedings of the OSDI. 10: 423-436. 2010.

[11] Fran Berman, Geoffrey Fox, and Anthony JG Hey. *Grid computing: making the global infrastructure a reality.* Journal of John Wiley and sons. 2: 217-249. 2003.

[12] A Boukerche. *Handbook of algorithms for wireless networking and mobile computing.* Journal of CRC Press. Page: 2(5)-2(20).

[13] A Boukerche. *Algorithms and protocols for wireless, mobile Ad Hoc networks.* Journal of John Wiley and Sons. Page: 1-21, 2008.

[14] A Boukerche. *Algorithms and protocols for wireless sensor networks.* Journal of John Wiley and Sons. Page 21-51, 2008.

[15] A Boukerche and L Bononi. *Simulation and modeling of wireless, mobile, and ad hoc networks.* Journal of Mobile ad hoc networking. Page: 373-409, 2004.

[16] A Boukerche, I Chatzigiannakis, and S Nikoletseas. *A new energy efficient and fault-tolerant protocol for data propagation in smart dust networks using varying transmission range.* Journal of Computer communications 29 (4). Page: 477-489, 2006.

[17] A Boukerche, X Cheng, and J Linus. *A performance evaluation of a novel energy-aware data-centric routing algorithm in wireless sensor networks.* Journal of Wireless Networks 11 (5). Page: 619-635, 2005.

[18] A Boukerche and SK Das. *Dynamic load balancing strategies for conservative parallel simulations.* In proceedings of 11th Workshop onrallel and Distributed Simulation. Page: 20-28, 1997.

[19] A Boukerche, SK Das, and A Fabbri. *Analysis of a randomized congestion control scheme with DSDV routing in ad Hoc wireless networks.* Journal of Parallel and Distributed Computing 61 (7). Page: 967-995, 2001.

[20] A Boukerche, SK Das, and A Fabbri. *SWiMNet: a scalable parallel simulation testbed for wireless and mobile networks.* Journal of Wireless Networks 7 (5). Page: 467-486, 2001.

[21] A Boukerche, K El-Khatib, L Xu, and L Korba. *SDAR: a secure distributed anonymous routing protocol for wireless and mobile ad hoc networks.* In proceeds of Local Computer Networks. Page: 618-624, 2004.

[22] A Boukerche and X Fei. *A voronoi approach for coverage protocols in wireless sensor networks.* In proceeds of the Global Telecommunications Conference. Page: 5190-5194, 2007.

[23] A Boukerche and X Fei. *A coverage-preserving scheme for wireless sensor network with irregular sensing range.* Journal of Elsevier Ad hoc networks 5 (8), 1303-1316, 2007.

[24] A Boukerche, S Hong, and T Jacob. *An efficient synchronization scheme of multimedia streams in wireless and mobile systems.*

[25] A Boukerche, S Hong, and T Jacob. *A distributed algorithm for dynamic channel allocation.* Journal of Mobile Networks and Applications 7 (2). Page: 115-126, 2002.

[26] A Boukerche and MSM Annoni Notare. *Behavior-based intrusion detection in mobile phone systems.* Journal of Parallel and Distributed Computing 62 (9). Page: 1476-1490, 2002.

[27] A Boukerche, HAB Oliveira, EF Nakamura, and AAF Loureiro. *Secure localization algorithms for wireless sensor networks.* Journal of IEEE Communications Magazine, 46 (4). Page: 96-101, 2008.

[28] A Boukerche, HABF Oliveira, EF Nakamura, and AAF Loureiro. *Vehicular ad hoc networks: A new challenge for localization-based systems.* Journal of Computer communications 31 (12). Page: 2838-2849, 2007.

[29] A Boukerche, HABF Oliveira, EF Nakamura, and AAF Loureiro. *Localization systems for wireless sensor networks.* Journal of IEEE Wireless Communications, 14 (6), page: 6-12, 2007.

[30] A Boukerche, RWN Pazzi, and RB Araujo. *Hpeq a hierarchical periodic, event-driven and query-based wireless sensor network protocol.*

[31] A Boukerche and Y Ren. *A secure mobile healthcare system using trust-based multicast scheme.*

[32] A Boukerche and A Roy. *Dynamic grid-based approach to data distribution management.* Journal of Parallel and Distributed Computing 62 (3). Page: 366-392, 2002.

[33] A Boukerche and C Tropper. *A distributed graph algorithm for the detection of local cycles and knots.* Journal of IEEE Transactions on Parallel and Distributed Systems, 9 (8). Page: 748-757, 1998.

[34] Azzedine Boukerche and Robson Eduardo De Grande. *Dynamic load balancing using grid services for hla-based simulations on large-scale distributed systems.* In proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications. Page: 175–183. 2009.

[35] Buyya, Rajkumar, Chee Shin Yeo, and Srikumar Venugopal. *Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities.* In proceedings of the High Performance Computing and Communications, HPCC'08. Page: 5–13. 2008.

[36] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. *Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities.* In proceedings of the High Performance Computing and Communications. Page: 5-13. 2008.

[37] Wentong Cai, Stephen John Turner, and Hanfeng Zhao. *A load management system for running HLA-based distributed simulations over the grid.* In proceedings of Distributed Simulation and Real-Time Applications. Page: 7–14. 2002.

[38] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, AF De Rose, and Rajkumar Buyya. *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.* Journal of Software: Practice and Experience 41(1): 23-50. 2011.

[39] Centos. *Centos Images.* Available: http://cloud.centos.org/centos/6/images/. Accessed on January 6, 2015.

[40] Nick Collier. *Repast: An extensible framework for agent simulation.* Journal of the University of Chicagos Social Science Research. 36: 2003. 2003.

[41] Judith S. Dahmann, Richard M. Fujimoto, and Richard M. Weatherly. *The department of defense high level architecture.* In proceedings of the 29th conference on Winter simulation. Page: 142–149. 1997.

[42] Robson Eduardo De Grande and Azzedine Boukerche. *Predictive dynamic load balancing for Large-Scale HLA-based simulations.* In proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications. Page: 4–11. 2011.

[43] Robson Eduardo De Grande and Azzedine Boukerche. *Distributed dynamic balancing of communication load for large-scale HLA-based simulations.* In proceedings of the Computers and Communications (ISCC). Page: 1109–1114. 2010.

[44] H. de Oliveira, A Boukerche, E Freire Nakamura, and AAF Loureiro. *An efficient directed localization recursion protocol for wireless sensor networks.* Journal of IEEE Transactions on Computers, 58 (5). Page: 677-691, 2009.

[45] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: simplified data processing on large clusters.* Journal of Communications of the ACM. 51(1): 107-113. 2008.

[46] M Elhadef, A Boukerche, and H Elkadiki.

[47] Fedora. *Fedora Project.* Available: http://torrents.fedoraproject.org/. Accessed on January 6, 2015.

[48] Shaochong Feng, Yanqiang Di, and Zhu Xianguo Meng. *Remodeling traditional rti software to be with paas architecture.* Journal of Computer Science and Information Technology (ICCSIT). 1: 511-515. 2010.

[49] and Kesselman C. Foster, I., J.M. Nick, and S Tuecke. *Grid services for distributed system integration.* Journal of Computer. 35(6): 3746. 2002.

[50] I. Foster, C. Kesselman, and S Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* Journal of High Performance Computing Applications. 15(3): 200–222. 2001.

[51] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a new computing infrastructure.* Journal of Elsevier. Page: 1–7. 2003.

[52] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. *Cloud computing and grid computing 360-degree compared.* In proceedings of the Grid Computing Environments Workshop, GCE'08. Page: 1-10. 2008.

[53] Richard M Fujimoto. *Parallel and distributed simulation systems.* Journal of Wiley New York. Page 3–8. 2000.

[54] Richard M Fujimoto, Asad Waqar Malik, and Alfred J Park. *In proceedings of Parallel and distributed simulation in the cloud.* Journal of SCS M&S Magazine. 3: 1-10. 2010.

[55] Garg, Saurabh Kumar, and Rajkumar Buyya. *Networkcloudsim: Modelling parallel applications in cloud simulations.* In proceedings of the Utility and Cloud Computing (UCC). Page: 105–113. 2011.

[56] Heng He, Ruixuan Li, Xinhua Dong, Zhi Zhang, and Hongmu Han. *An Efficient and Secure Cloud-Based Distributed Simulation System.* Journal of Applied Mathematics & Information Sciences. 6(3): 729-736. 2012.

[57] Hofer, Ronald C, and Margaret L Loper. *DIS today [Distributed interactive simulation].* In proceedings of the IEEE 83(8): 1124-1137. 1995.

[58] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. *Performance analysis of high performance computing applications on the amazon web services cloud.* In proceedings of the Cloud Computing Technology and Science (CloudCom). Page: 159–168. 2010.

[59] Karl Jones. *Karl Distributed Interactive Simulation.* Available: http://sourceforge.net/projects/kdis/. Accessed on January 6, 2015.

[60] In-Yong Jung, Byong-John Han, and Chang-Sung Jeong. *Provisioning On-Demand HLA/RTI Simulation Environment on Cloud for Distributed-Parallel Computer Simulations.* In proceedings of the Mobile, Ubiquitous, and Intelligent Computing. Page: 329-334. 2014.

[61] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. *kvm: the Linux virtual machine monitor.* In proceedings of the Linux Symposium. 1: 225–230. 2007.

[62] Bo Hu Li, Xudong Chai, Baocun Hou, Chen Yang, Tan Li, Tingyu Lin, Zhihui Zhang, Yabin Zhang, Wenhai Zhu, and Zenghui Zhao. *Research and application on cloud simulation.* In proceedings of the 2013 Summer Computer Simulation Conference. Page: 34:1 - 34:14. 2013.

[63] Xiaocheng Liu, Xiaogang Qiu, Bin Chen, and Kedi Huang. *Cloud-based Simulation: the State-of-the-art Computer Simulation Paradigm.* In proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation. Page: 71-74. 2012.

[64] Asad Waqar Malik, Alfred Park, and Richard M Fujimoto. *Optimistic synchronization of parallel simulations in cloud computing environments.* In proceedings of the Cloud Computing, CLOUD '09. Page: 49-56. 2009.

[65] Miller, Duncan C., and Jack A. Thorpe. *SIMNET: The advent of simulator networking.* In proceedings of the IEEE. 83(8): 1114-1123. 1995.

[66] HABF Oliveira, A Boukerche, EF Nakamura, and AAF Loureiro. *Localization in time and space for wireless sensor networks: An efficient and lightweight algorithm.* Journal of the Performance Evaluation 66 (3). Page: 209-222, 2009.

[67] Object Management Group (OMG). *Data Distribution Service for Real-time Systems.* Available: http://www.omg.org/spec/DDSII.2. Accessed on January 6, 2015.

[68] OpenDDS. *Introduction to OpenDDS.* Available: http://www.opendds.org/. Accessed on January 6, 2015.

[69] George Pallis. *Cloud Computing, The New Frontier of Internet Computing.* Journal of Cloud Computing. Page: 1–8. 2010.

[70] Pardo-Castellote and Gerardo. *Omg data-distribution service: Architectural overview.* In proceedings of the Distributed Computing Systems Workshops. Page: 200–206. 2003.

[71] Ken Pepple. *Deploying OpenStack.* Journal of O'Reilly Media, Inc. Page: 1–10. 2011.

[72] poRTIco. *The poRTIco Project.* Available: http://www.porticoproject.org/. Accessed on January 6, 2015.

[73] PRESAGIS. *Pitch HLA for STAGE.* Available: http://www.presagis.com/products-services/products/options/pitch/. Accessed on January 6, 2015.

[74] PRISMTECH. *Vortex OpenSplice.* Available: http://www.prismtech.com/vortex/vortex-opensplice. Accessed on January 6, 2015.

[75] Y Ren and A Boukerche. *Modeling and managing the trust for wireless and mobile ad hoc networks.* In proceedings of Communications, ICC'08. Page: 2129-2133, 2008.

[76] Katarzyna Rycerz. *A framework for HLA-based interactive simulations on the grid.* Journal of Simulation. 81(1): 67-76. 2005.

[77] S Samarah, M Al-Hajri, and A Boukerche. *A predictive energy-efficient technique to support object-tracking sensor networks.* Journal of the Vehicular Technology, IEEE Transactions on 60 (2). Page: 656-663, 2011.

[78] Dan Sanderson. *Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure.* Journal of O'Reilly Media, Inc. Page: 1–10. 2009.

[79] SIMWARE. *SimWare RTI.* Available: http://www.simware.es/index.php/simware-products/products-simware-rti. Accessed on January 6, 2015.

[80] S. I. S. C. (SISC). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Framework and Rules.* IEEE Computer Societ. 2000.

[81] Sotiriadis and Stelios. *SimIC: Designing a new Inter-Cloud Simulation platform for integrating large-scale resource management.* In proceedings of the Advanced Information Networking and Applications (AINA). Page: 90–97. 2013.

[82] Sriram and Ilango. *SPECI, a simulation tool exploring cloud-scale data centres.* Journal of Cloud Computing. Springer Berlin Heidelberg. Page: 381-392. 2009.

[83] Georgios Theodoropoulos, Yi Zhang, Dan Chen, Rob Minson, Stephen John Turner, Wentong Cai, Yong Xie, and Brian Logan. *Grid-aware large scale distributed simulation of agent-based systems.* In proceedings of the Cluster Computing and the Grid, Cluster Computing and the Grid, CCGRID 06. 2: 63. 2006.

[84] Ryan Troy and Matthew Helmke. *VMware Cookbook: A Real-World Guide to Effective VMware Use.* Journal of O'Reilly Media, Inc. Page: 3–10. 2012.

[85] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. *A break in the clouds: towards a cloud definition.* Journal of ACM SIGCOMM Computer Communication Review. 39(1): 50-55. 2008.

[86] David Villegas, Ivan Rodero, Liana Fong, Norman Bobroff, Yanbin Liu, Manish Parashar, and S Masoud Sadjadi. *The role of grid computing technologies in cloud computing.* Journal of the Handbook of Cloud Computing. Page: 183-218. 2010.

[87] Bill Wilder. *Cloud Architecture Patterns: Using Microsoft Azure.* Journal of O'Reilly Media, Inc. Page: 1–5. 2012.

[88] Yong Xie, Yong Meng Teo, Wentong Cai, and Stephen John Turner. *Service provisioning for HLA-based distributed simulation on the Grid.* In proceedings of the

Principles of Advanced and Distributed Simulation, PADS 2005. Page: 282–291. 2005.

[89] Jinzy Zhu, Xing Fang, Zhe Guo, Meng Hua Niu, Fan Cao, Shuang Yue, and Qin Yu Liu. *IBM cloud computing powering a smarter planet.* Journal of Cloud Computing. Page: 621–625. 2009.