Enabling HLA-based Simulations on the Cloud †

Shichao Guan
PARADISE Research Laboratory
EECS - University of Ottawa
Email: sguan049@uottawa.ca

Robson Eduardo De Grande
PARADISE Research Laboratory
EECS - University of Ottawa
Email: rdgrande@site.uottawa.ca

Azzedine Boukerche
PARADISE Research Laboratory
EECS - University of Ottawa
Email: boukerch@site.uottawa.ca

***Abstract*—The HLA framework is widely used to formalize simulations and achieve reusability and interoperability of simulation components. In order to manage the underlying system of HLA-based simulations, Grid Computing and Cloud Computing are employed to tackle the details of operation, configuration, and maintenance of simulation platforms that simulation applications run on. However, to make a simulation-run-ready environment among different types of computing resources and network environments is challenging, especially for modelers who may not be familiar with the management of distributed systems. In this article, we propose a new cloud-based scheme for HLA-based simulations, aiming to ease the management of underlying resources, particularly for those located on geographically distributed locations, and to achieve rapid elasticity that can provide adequate computing capability to end users. An approach for handling diverse network environments is given; by adopting it, idle public resources can be easily configured as additional computing resources for the local cloud infrastructure. In the experiments, compared with its corresponding Grid Computing platform, this Cloud Computing platform achieves a similar performance but with many advantages that Cloud can provide, such as energy consumption, security, and multi-user availability.**

***Keywords*-Cloud Computing; HLA; Availability; Usability; Distributed Simulations.**

I. Introduction

The High Level Architecture (HLA) developed by the Department of Defence in the United States is a framework that provides reusability and interoperability for distributed simulations. By utilizing this framework, modeling and simulation in analysis, engineering, military, entertainment, education, and a variety of other applications can be linked to live systems, collectively defined as federates.

HLA has been developed as an IEEE standard [1] for modeling and simulation. Coordinated by this standard, large-scale distributed simulations can be deployed to geographically distributed computing resources. However, within this standard, mechanisms for executing simulations are not provided for the underlying systems. For each physical computation element, it is necessary to carefully create a series of configurations before the HLA Run-Time Infrastructure (RTI) middleware is able to work properly.

In order to meet the need for managing the underlying system, Grid Computing [2] is introduced to manage shared resources and the scheduling HLA simulations. Grid performs as a coordinated resource sharing system that provides services such as security, resource management, information queue, and data management for distributed simulations. The Globus Toolkit (GT) by Foster [3] is defined as a de-facto middleware standard for Grid Computing that helps to reach seamless interoperability. With the help of Grid services [4], Grid-based simulation platforms, compared with traditional compute simulation systems, overcome limitations in terms of distribution of resources, dynamic access, security, organization, and collaboration.

Even though Grid eases the management of underlying systems for distributed simulations, it is hard to handle resources in a fine-grained manner. As a result, Grid-based platforms cannot reallocate resources during run-time, and support multi-users fully. To tackle these issues with the underlying system, Cloud Computing [5] is employed as a new approach for distributed simulations. According to the analyses in [6], [7], [8], and [9], there are several reasons to migrate distributed simulations from Grid to Cloud:

- Resource sharing: Cloud provides resources on demand during run-time while Grid emphasizes fair share of resources across organizations.
- Virtualization: For Grid, virtualizaion mainly covers softlayer – that is, data and programming. For Cloud, in addition to the softlayer, virtualization covers hardware resources. By enabling abstraction and encapsulation of row resources, Cloud can provide better isolation and manageablility for fine-grained resources.
- Scalability: Grid scales primarily by increasing the number of working nodes. Cloud offers automatic resizing of virtualized hardware.
- Payment: Grid services are typically billed using a fixed rate, while Cloud users can use a pay-per-use flexible model.

Although Cloud Computing has not been standardized to inner interfaces for accessing resources, organizations such as OCCI group have made great efforts to unify public cloud interfaces. Compared with these Grid services, the Cloud services – 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, incorporating different

† This work is partially supported by NSERC, the Canada Research Chair program, MRI/ORF research funds, and the Ontario Distinguished Research Award.

aspects in their design, such as job scheduling, monitoring, and security. However, these existing cloud platforms are implemented based mainly 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 Cloud Computing's essential characteristic of elasticity (according to the definition from the National Institute of Standards and Technology (NIST) [10]).

In this paper, a new cloud simulation scheme is proposed that is capable of providing adequate computing resources to end users by coordinating public compute resources during run-time. As a cloud simulation platform, it also contains components for job scheduling, monitoring, and a friendly web-based graphic interface to ease the configuration, operation, and maintenance of the underlying system.

The reminder of the paper is organized as follows. In Section 2, the state of the art research is shown and discussed. In Section 3, the proposed scheme is described with the architecture, key components and functioning. In Section 4, several experiments are deployed and discussed in terms of energy consumption, performance, and results. In the last section, the conclusion summarizes the proposed solution design and the experimental results, and future directions are presented.

II. Related Work

Because of the novelty and benefits of Cloud Computing, many Cloud-based platforms are proposed for distributed simulations, focusing on on-demand resource provisioning, virtualization management, security, monitoring, optimization of data processing, synchronization, etc. These platforms mainly concentrate on simulations running on local resources, but they do not consider situations in which local resources are exhausted – the ability of rapid elasticity during run-time.

In terms of execution efficiency, the Aurora system [11] and the Time Warp Straggler Message Identification Protocol (TW-SMIP) [12] are introduced to tackle issues in small message communication and optimistic synchronization. In the Aurora system [11], a master/worker paradigm is introduced to aggregate small communication messages for different destinations. These messages are automatically bundled and transmitted as one unit. By doing so, high bandwidth and high latency networks are better utilized in cloud environments. In the TW-SMIP, heartbeat messages are introduced to detect straggler messages. The straggler message identification is utilized to reduce the amount of rollbacks that may be caused by asymmetric or uneven processing loads. In addition, in this paper, three types of SMIP protocols are proposed to distribute heartbeat messages that concern congestion, autonomous administration and communication overhead respectively. Cloud-based Simulation (CSim) [13] concerns the idle CPU cycle issue. In this proposed scheme, each processor is virtualized as two CPUs (VCPU) – one foreground and one background. 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. In the Cloud-based Distributed Simulation system (CDS) [14], a new Cloud-RTI middleware based on traditional RTI is proposed. With the encapsulation of 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, the Cloud Simulation System in [19] 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 behaviours. In [14], a hierarchical identity-based cryptography and a role-based access control scheme are deployed. In this scheme, the root Key Generation Center (KGC), domain KGC, 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.

In [15], a communication architecture is implemented, using Software Defined Networks technology to tackle Data Distribution Service related applications, achieving more efficiency in terms of network resource usage.

In [16], a two-layered grid-based system is designed, aiming at reducing power consumption. Power saving mechanisms are proposed and evaluated with simulations.

Many Cloud platforms attempt to draw a full-scale picture of Cloud-based simulation, for example, [17], [19], [18], [13], [20], and [14]. In [19], a four-layered platform is implemented. In this platform, several technologies are proposed that focus on different aspects of distributed simulations. A management system is created for various resources. Individuation virtual desktop technology, multi-user oriented dynamic building technology, and automatic composition 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 sub-model issue in one federate. Monitoring and evaluation technology is implemented for detecting abnormal behaviours on the platform and providing information for further optimization decisions. In [17] and [14], performance comparisons are made between simulations of cloud virtual instances and simulations on native hosts. In [20], time-latency is compared between Cloud-based RTI and native Portico RTI [21] and it shows that, within a certain range of length of update data, time-latency is acceptable on the cloud.

Even though the aforementioned approaches show many gains in certain aspects, based on the descriptions of their experiments, only limited local physical resources are virtualized and coordinated in the cloud resource pool. In this case, to end users, the compute utility seems inadequate, particularly for large-scale distributed simulations. In this paper, we proposed a new elastic cloud simulation scheme that is able to scale outwards rapidly depending on the demands of the end user. The compute capability is increased by introducing public hosts to the local cloud resource pool during run-time. These public hosts include physical desktops, workstations, laptops, or public virtual cloud instances from different cloud providers, such as AWS, Google Compute Engine, and Azure,

which can be located in geographically separate regions. On this platform, several components are designed to manage the public resources run-time provisioning. These components make the public hosts perform as they are in the same local LAN, even though they are in different locations. This is significant for some HLA-based RTI middlewares, which use multicast or other specific transmission methods to exchange information among federates.

III. Cloud Simulation Platform

A layered platform is designed to ease the management of the underlying systems for modelers. New components, such as physical resources, softlayer data and programs, can be handled by the platform as elastic, plugged-in components, requiring few configurations. By controlling the granularity of both hardlayer and softlayer resources, the cloud platform achieves better flexibility. A web-based secure graphic portal makes it possible for users to access sufficient computing resources through lightweight terminals.

A. Architecture

This cloud Simulation platform, as shown in Figure 1, consists of four layers: Row Resource Layer, Integration and Virtualization Layer, Simulation Function Layer, and Modeler Management Layer.

- Modeler Management Layer: This layer is presented as a portal to the user. Concerning security issues, layered authentication methods are implemented to protect cloud core data and user sensitive information.
- Simulation Function Layer: This layer mainly contains three parts: the Simulation Resource Manager, the Simulation Resource Pool, and the Virtual Resource Scheduler. Above them, two types of interfaces are proposed for users to access core simulation functions. A web-based simulation workbench is provided, on which diverse simulation experiments can be designed. Services are available for modelers to code, schedule jobs, execute simulations, analyse results, and share experiences.
- Integration and Virtualization Layer: This layer includes the local Virtual Resource Manager and the Public Resource Agent. In this layer, row resources are virtualized and integrated, performing as one united multi-processor system.
- Row Resource Layer: This layer contains a local row resource pool which includes computing resources, storage resources, and network resources. In this layer, physical hosts only contain an OS and basic software that such an OS brings.

B. Key Features

In this subsection, five key components of this cloud simulation platform, as shown in Figure 1, are explained in detail to show the core functioning of the system: the User Portal in Modeler Management Layer; the Simulation Resource Manager and the Virtual Resource Scheduler in Simulation Function Layer; and the Virtual Resource Manager and the Public Resource Agent in Intergration and Virtualization Layer.

1) User Portal: Concerning security issues on the cloud platform, the User Portal is introduced to provide authentication and access control. In this layer, a password is used to authenticate the user's identity, and a user-based certification, as well as an X-token, secures access to compute servers and virtual instances respectively. The certification is required when users attempt to link to the Simulation Resource Pool, as shown in the Simulation Function Layer. With this certification provided, users are able to acquire plenty of simulation resources provided by the cloud simulation platform. The X-token is an access key for the user to manage its personal virtual instances in the Virtual Resource Pool, as shown in the Simulation Function Layer. The X-token is required when users create, access, or delete instances. With the access control and virtualization technology, users on the cloud are better isolated so that operations from different users on the same physical host do not influence each other. At any time, one virtual instance can only belong to one cloud user for the exclusive usage. The GUI and command-line based interface are both implemented for users to create, execute and maintain computing resources and virtual computing capability.

2) Simulation Resource Manager: When modelers get through the aforementioned authentication, they tap into a web-based simulation workbench, within which they are able to design simulation applications online through only a lightweight terminal, as shown in Figure 1. In addition, users can take advantage of the Simulation Resource Pool that gathers and presents federations, federates, configuration files, templates, solutions, and applications shared among the cloud users. This Simulation Resource Pool is managed by the Simulation Resource Manager, which also plays a core role in the background storage system. The Simulation Resource Manager provides two types of storage: cloud storage and local storage. The former is mainly used for replication to backup resources in the Simulation Resource Pool, in the event that local resource storage are temporarily unavailable. The latter stores all necessary information that this cloud simulation platform requires during run-time and it utilizes the Network File System (NFS) to accelerate the scheduling and execution.

3) Virtual Resource Scheduler: Due to the diversity of HLA-based modeling and simulation applications, requirements of computing resources differ. The Virtual Resource Scheduler is utilized to control the granularity of resources so that the needs of different simulation applications can be met. For end users, they can define simulation environment parameters such as the number of virtual processors, memory size, network topology, and the quantity of virtual machines, according to the real needs of simulations. The Virtual Resource Scheduler is responsible for handling these requests, calling relevant components in the local Virtual Resource Manager and the Public Resource Agent to initiate virtual instances accordingly in the integrated and virtualized computing resource pool.

Several Scheduling algorithms are provided to initiate vir-

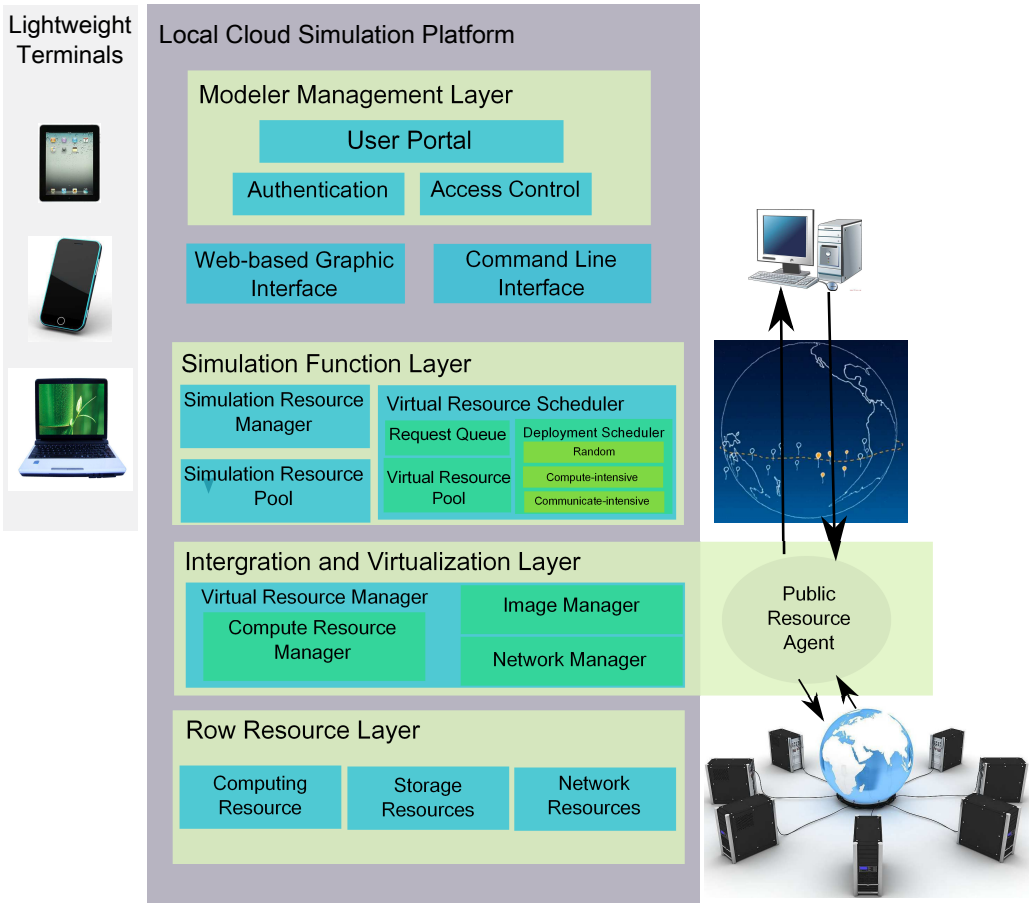

Fig. 1. Cloud Simulation Scheme

tual instances, as shown in Algorithm 1: random, computation-intensive, and communication-intensive. N in line 1 is the number of virtual resources that require deployment in simulations. The default value is 0. 0 means that the current available resources are already adequate and no additional virtual resources are to be scheduled from the Virtual Resource Pool. In this context, IT in line 1 is short for the instance type. This parameter defines the characteristics of virtual instances that will be further created and utilized. It contains compute capability information such as processors, RAM, and hard disk. SA in line 1 stands for the scheduling algorithms. In Random, as shown from line 4 to line 12, the Virtual Resource Scheduler randomly creates instances. First, from the Virtual Resource Pool, a queue of all available resources is created. Then, based on the length of this queue, a random number n is created. The resource whose index number is n in the queue is selected as the destination resource. After the selection, the request to initiate virtual instances is sent to the destination resource boot queue, ready for creation. Finally, the system records and updates the resource pool metadata before the next round of scheduling starts. In the computation-intensive section, in lines 13 to 20, resources in the available queue are first sorted based on their current available computing capability. Then, the resource with the smallest index number is selected as the destination resource. With this scheduling algorithm, high performance computing virtual instance type is often selected as IT to suit simulations which involve a large computation load but a small communication load. In communication-intensive mode, shown between lines 21 and 29, a queue of all available resources is listed, which is the same as the random scheduling. Then, the first resource in the queue is selected as the destination resource. This resource attempts to boot as many virtual instances as possible until its computing capability is exhausted. After one resource is used up, the next resource in the queue becomes the target destination resource. In this way, communication-intensive federates can be better initiated in the same physical host to reduce communication distance.

4) Virtual Resource Manager: This plays a central role in integrating local resources. The local Virtual Resource Manager contains three sub-managers: Compute Resource Manager, Image Manager, and Network Manager, which deal with the type of instances, the OS and software of instances, and the local instance network topology respectively. The Network Manager also controls a key daemon that manages communications between local virtual instances and public instances. This daemon monitors the virtual tunnels that are built by the Public Resource Agent, mapping local instances

Algorithm 1: Scheduling Algorithm

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

Algorithm 2: Access Control & Initialization Algorithm

```
1  Require: username/password, private_key, x_token
2  if get_user_pass() then
3      current_pri ⇐ pri_comparison(current_pri, r_only)
4      if get_private_key() then
5          current_pri ⇐ pri_comparison(current_pri, editable)
6          if get_x_token() then
7              current_pri ⇐
               pri_comparison(current_pri, executable)
8          else
9              current_pri ⇐ editable
10     else
11         current_pri ⇐ r_only
12 else
13     current_pri ⇐ NONE
14 if current_pri >= editable then
15     Initialize :
16     N ⇐ 0
17     IT ⇐ def(small)
18     SA ⇐ def(co − int)
19     P ⇐ def(pack_list)
20     PL ⇐ def(key_list, ip_list)
```

to the proper public tunnel. In order to support multi-user and reduce energy consumption, virtual instances are deployed during run-time based on the requests from the aforementioned Virtual Resource Scheduler. Once the virtual instances are destroyed, the computing capabilities are recycled in the local resource pool for further usage.

5) Public Resource Agent: In order to build an elastic platform containing both local resources and public resources, the diversity of public network environments should be taken into consideration. In this environment, some protocols such as multicast and broadcast may not be well supported by routers, switches or hubs. For example, routers drop multicast messages by default, if the destination IP and the resource IP are not in the same subnet. For this reason, the Public Resource Agent is introduced to manage the public network. For each public instance, the Public Agent virtualizes a VPN-based tunnel over the physical network, penetrating the public Internet and encapsulating packages and messages that need to be sent and received. Meanwhile, related Firewall rules, routing tables and ARP tables in the public instance are automatically modified, allowing forwarding and receiving packages and messages via the virtual tunnel. To ensure better simulation performance, public instances are added to the local resource pool only when local computing resources are not sufficient to reduce network latency.

C. Cloud Simulation Scheme

As described in Algorithm 2 and Algorithm 3, the cloud simulation scheme consists mainly of two parts and four phases: Access Control, Initialization, Virtual Resource Scheduling, and Simulation Execution. These phases cover typical scenarios for HLA-based simulations.

In Access Control, three types of priorities are given based on the user's role. r_only in line 3, Algorithm 2 is the lowest priority that only requires password as authentication, allowing users to review simulation resources, such as templates, shared federates, and shared documents in the Simulation Resource Pool. $editable$ in line 5, in addition to r_only, provides the ability to modify, create and maintain simulation resources on the cloud platform. In order to obtain this level of priority, the private keys to relevant host services are required. Users whose priority is $executable$ in line 7 are able to tap into the virtual resource pool, initiate instances, execute simulations, analyse simulation results and share with self-defined groups. An x-token managed by cloud middleware is required for the designation of this level of priority. This token expires within 24 hours by default, and one new token can be acquired if necessary. In this case, if one user is attacked and loses control of the instances, an x-token reset command is able to block the hacker's access, preventing further destruction. By introducing Access Control, resources in the cloud simulation platform are more effectively utilized, and sensitive user data is protected, thus preventing malicious attacks inside the cloud securely.

After login, several parameters are initialized for users, as shown from lines 16 to 20 in Algorithm 2. These parameters initially set to default values, and can be modified to build self-defined simulation environments later. These self-defined simulation environments can be very flexible, even modifiable during run-time. As already mentioned in Algorithm 1 line 1, N, IT and SA are managed by the Virtual Resource Scheduler. N is set to 0 as initial value. P contains softlayer packages which are necessary for HLA-based simulations. PL represents the public instance list which contains the public host IP address and one access key to remotely control that public instance.

In the Virtual Resource Scheduling algorithm shown in

Algorithm 3: Virtual Resource Scheduling & Simulation Execution Algorithm

```
1  while !sche(end)&&get_pri >= executable do
2      while get_current_r <= get_req_r do
3          if get_req_r <= get_local_r then
4              N ⇐ get_loc_r − get_req_r
5              loc.b(N, get.u_def(IT), get.u_def(SA))
6              env.set(N, get.u_def(P))
7              r ⇐ update_req_r
8          else
9              N ⇐ get_req_r − get_local_r
10             loc.b(get_loc_r, get.u_def(IT), get.u_def(SA))
11             pub.set(N, get.u_def(PL))
12             env.set(N, get.u_def(P))
13             r ⇐ update_req_r
14     sche(end) ⇐ get_u.def(end)
15 while !sim(end)&&get_pri >= executable do
16     sim_vir_ins ⇐ get_u.def(ins_pool)
17     sim_code ⇐ get_u.upload(code_pool)
18     map_sim_pair(sim_vir_ins, sim_code)
19     sim_execute()
20     sim(end) ⇐ get_u.def(end)
21 if sim(end) then
22     sim_rec_rsl
23     sim_rel_res
24     current_pri ⇐ NONE
```

Algorithm 3, the user's priority to control virtual resources is first checked. If user-defined compute resources are not fully scheduled, the system then inspects the current local resource pool to determine whether it can sufficiently provide simulation computing capability in line 3. If current available local resources are adequate, instances should boot locally and the Public Resource Agent is not contacted. In this case, better performance can be achieved by avoiding communication through a long-distance public network. When the local resource pool is not enough for large-scale simulations as shown in lines 9 to 13, the Virtual Resource Manager first schedules all available local resources. Then, after these are exhausted, the Public Resource Agent is triggered and public instances are configured to elastically provide additional compute capability.

Based on the above three phases, the simulation environment is deployed properly according to the user's personal setting. During the last step before simulations can be executed, each federation and federate are mapped to the virtual instance it runs. The mapping is done from 16 to 18 in Algorithm 3. From the user's private virtual resource pool and simulation resource pool, a list of virtual machines and federations/federates are selected. Then, the mapping is done by pairing the index of selected simulation resources and the index of selected virtual resources. Simulation resources are delegated to destination virtual instances based on the index pair. At this time, the simulation is ready for execution. When one round of simulation completes, federates/federations can be quickly remapped to other virtual instances for comparison of the results. When simulation ends, its corresponding computing resources are recycled, and its simulation resources can be stored in the cloud.

TABLE I
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

IV. EXPERIMENTS AND RESULT ANALYSIS

The prototype of this platform has been implemented and deployed on a DELL cluster composed of 20 nodes. Each node in the Dell cluster contains a QuadiCore 2.40GHz Intel(R) Xeon(R) CPU and 8 gigabytes of DIMM DDR RAM memory, and all nodes are interconnected through a Myrinet optical network which allows data transmissions of up to 2 gigabytes per second. Experiments are devised to evaluate the system's energy consumption, resource provisioning time, and simulation performance based on different types of instances, scheduling algorithms, and simulation platforms.

The first experiment shows the execution time of local simulation environment preparation, which includes virtual instance boot time, compute resource scheduling time, and software deployment & simulation package delegation time. In the experiment, Linux System is used as the OS, and the simulation example uses one restaurant simulation application [21] from Protico java-based RTI. The virtual instance type in this experiment is shown in TABLE I.

The results of this first experiment are presented in Table II. Virtual instance scheduling time, instance booting time and package preparation time are calculated respectively. T_rare is the time required to boot a rare OS, which in this experiment means to a Fedora 17. As seen in the table, approximately only 17 seconds are spent on booting the OS by the virtual instances. T_sche is the time consumed by system scheduling, which begins when the user's resource provisioning request is accepted from a web-based interface, and ends after all these requests are properly stored, recorded, and processed in the Request Queue. T_pac calculates the total amount of time for preparing a simulation-run-ready environment, which includes the deployment of OS, simulation source code, simulation packages, and simulation depended packages. The results indicate that within three minutes, the simulation environment is completely up and ready for executing the simulations. It is worth mentioning that the most time-consuming aspect is the deployment of softlayer support packages required for simulations.

The second experiment concerns the performance of three types of scheduling algorithms when simulations are scaled up. In this experiment, mini virtual instances and large virtual instances are introduced as virtual instance examples. These instances are scheduled until the compute capability of local resources reaches 75 percent. For the random scheduling algorithm and the communication-intensive scheduling algorithm, mini virtual instances are employed to benefit these

TABLE II
ENVIRONMENT PREPARATION TIME CONSUMPTION

Virtual Resource Type	T_rare(s)	T_sche(s)	T_pac(s)
Mini	16.871	1.389	153.405
Small	17.278	1.421	139.234
Medium	17.502	1.322	138.071
Large	17.892	1.433	147.977

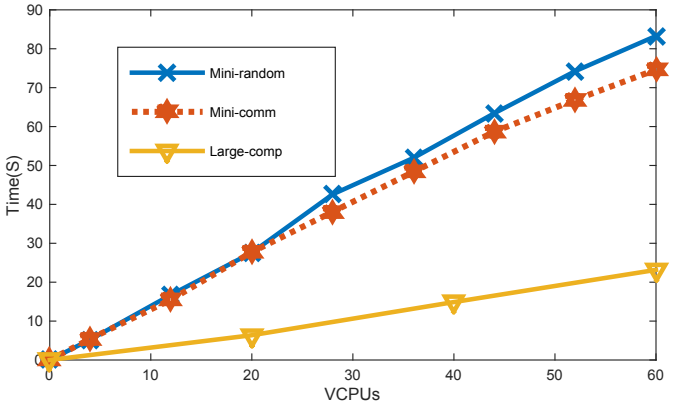

Fig. 2. Scheduling Algorithm Performance Evaluation

communication-intensive tasks. Large virtual instances, which are designed for computation-intensive work, are used as examples for computation-intensive scheduling algorithms. The scheduling time recorded in this experiment is the same as T_sche in Table II.

As depicted in Figure 2, with simulations are scaled up, the scheduling time has a nearly linear increase rate, except for some slight fluctuations caused by the background load. The background load originates from the cloud system, which needs to keep running its services for periodical detection and status updates of the physical machines that compose the environment. Such services coordinate the services by observing and recording their power status, network availability, current free RAM, and other characteristics. Computation-intensive scheduling provides the best performance because it minimizes the rounds of resource scheduling. As for random and communication-intensive scheduling, the latter presents a superior overall performance by 10 percent. This is because, in the communication-intensive scheduling, the search queue for available resources in the resource queue is shorter – the first available physical host in the resource queue is selected as the destination host. In random scheduling, the average search length is $que.len/2$, which is much longer than communication-intensive scheduling. In addition, for communication-intensive scheduling, an optimization for cutting resource recording time is utilized. Communication-intensive scheduling records only when one physical host is exhausted, while random scheduling must record immediately after the end of each scheduling round. For the different types of scheduling algorithms used in this platform, the scheduling time for one instance is less than 1.5 seconds.

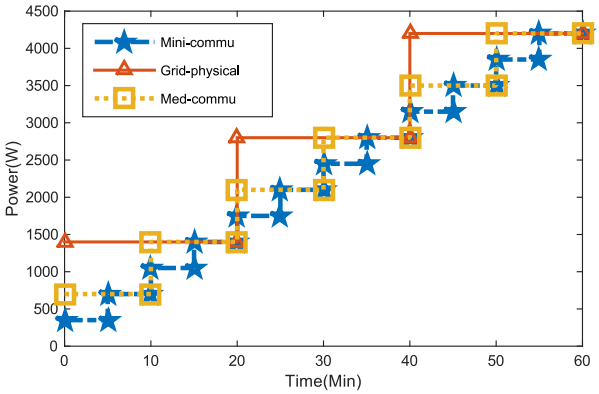

Fig. 3. Energy Consumption between Cloud Simulation Platform and Grid Platform

Instead of manually configuring relevant simulation aspects, the automatic scheduling components of this platform can save modelers plenty of time.

The third experiment focuses on the aspect of energy consumption in this scheme. In the experimental simulation scenario, we suppose that four users would gradually exhaust the compute capability of twelve physical machines in the cluster within an hour. In terms of security, each user has exclusive access to the resources that the user operates. For the cloud scenario, one new VCPU and its related computing capability is consumed by each user every five minutes. For a grid scenario using native physical machines, one new CPU and related computing capability is consumed by each user during the same time period as the cloud. When sixty minutes has elapsed, a total of 48 VCPUs or CPUs with corresponding compute capability are running in twelve machines. The energy consumption is calculated for the cloud platform and the native grid system respectively based on the current rated power of the physical machine, which is 355 W in this experiment. Communication-intensive scheduling is used to boot instances for the cloud platform and the grid platform.

From the results depicted in Figure 3, we can see that the Cloud simulation platform conserves more energy than its Grid counterpart. Compared with physical hosts, small virtual instances on the cloud are more agile and flexible, which can provide better resource utilization.

The last experiment is deployed to test the HLA-based simulation execution performance between the cloud simulation platform and the native grid simulation platform. In this experiment, a large virtual resource type (as shown in Table II) is used to boot virtual instances on the cloud, and grid particularly uses physical hosts directly. One sample federate using Protico java-based RTI [21] is deployed on fifteen virtual instances for the cloud, and fifteen physical hosts for the grid. This sample federate produces a controlled synthetic load, which consists of first creating a pseudo-random number x and then introducing a recursive procedure for x turns to exhaust compute capability for the instance in which it runs. After the computation, the sample federates communicate with

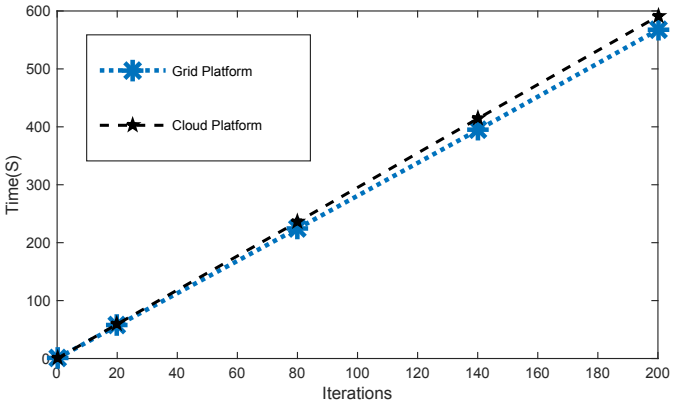

Fig. 4. Simulation Performance between Cloud Platform and Grid Platform

each other about their computation results. Different iterations are utilized to measure the simulation execution performance among fifteen large virtual instances on the Cloud and fifteen physical machines on the Grid.

According to Figure 4, the overall simulation performance of Grid outperforms its Cloud counterpart. When the experiment reaches 200 iterations, the performance of Grid is 3.94% better than Cloud. This difference in performance is caused due to the virtualization technology that is widely used to manage row resources for Cloud, which may lead to performance loss when compared with the performance of the native environment. This loss may vary depending on applications and execution contexts [22].

Based on the aforementioned four experiments, we conclude that our proposed system successfully provides elastic and automatic management of underlying resources. The configuration of resources is based on the requirements defined by users when deploying and running distributed simulations on the cloud, especially for HLA-based simulations. There is a loss of performance in the execution of simulations on this platform when compared with the native Grid platform, mainly because of the virtualization technology of the Cloud. However, this overhead is acceptable due to the benefits of Cloud Computing listed in the early sections of this paper.

V. Conclusion

In this paper, instead of utilizing cloud simulation tools, a real cloud-based simulation platform is presented and evaluated. This platform is implemented to ease the management of underlying resources for HLA-based simulations, and to achieve elasticity. Based on its layered design, users can easily access this platform through lightweight terminals and perform simulations on the provided workbench with an elastic resource pool. Several security approaches are proposed to protect sensitive information. Different types of scheduling and computing resources are introduced to meet the requirements of diverse simulation scenarios. Based on the experiments, this cloud scheme achieves a similar performance to its corresponding native grid platform, but it presents security for users, better system management, and less energy consumption.

References

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

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

[3] *Globus Toolkit*. University of Chicago. 7 Feb. 2008. Accessed on January 6, 2015.

[4] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke. *Grid services for distributed system integration.* IEEE Computer, 35(6): 37 - 46, 2002.

[5] A. Michael, A. Fox, R. Griffith, A. D. Joseph, R.Katz, A. Konwinski, G.Lee, P.David, R.Ariel, S.Ion, M. Zaharia. *A view of cloud computing*. Journal of the Communications of the ACM, 53(4): 50-58, 2010.

[6] I. Foster, Y.Zhao, I.Raicu, S.Lu. *Cloud computing and grid computing 360-degree compared.* In proc. of the Grid Computing Environments Workshop, GCE'08, page: 1-10, 2008.

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

[8] R. Buyya, C. S. Yeo, S. Venugopal. *Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities.* In proc. of the High Performance Computing and Communications, pp. 5-13, 2008.

[9] D. Villegas, I. Rodero, L. Fong, N. Bobroff, Y. Liu, M. Parashar, S. M. Sadjadi. *The role of grid computing technologies in cloud computing.* In Handbook of Cloud Computing, Springer, pp. 183-218, 2010.

[10] P. Mell, T. Grance. *The NIST definition of cloud computing.* National Institute of Standards and Technology, 53(6): 50, 2009.

[11] R. M. Fujimoto, A. W. Malik, A. Park. *Parallel and distributed simulation in the cloud.* Journal of the SCS M&S Magazine, 3: 1-10, 2010.

[12] A. W. Malik, A. Park, R. M. Fujimoto. *Optimistic synchronization of parallel simulations in cloud computing environments.* In proc. of the Cloud Computing, pp. 49-56, 2009.

[13] X. Liu, X. Qiu, B. Chen, K. Huang. *Cloud-based Simulation: the State-of-the-art Computer Simulation Paradigm.* In proc. of the ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation, pp. 71-74, 2012.

[14] H. He, R. Li, X. Dong, Z. Zhang, H. Han. *An Efficient and Secure Cloud-Based Distributed Simulation System.* Journal of the Applied Mathematics & Information Sciences, 6(3): 729-736, 2012.

[15] L.Bertaux, A.Hakiri, S.Medjiah, P.Berthou, S.Abdellatif. *A DDS/SDN based communication system for efficient support of dynamic distributed real-time applications.* In proc. of the IEEE/ACM 18th International Symposium onDistributed Simulation and Real Time Applications (DS-RT), pp. 77 - 84, 2014.

[16] G.Terzopoulos, H.D.Karatza. *Maximizing performance and energy efficiency of a real-time heterogeneous 2-level grid system using DVS.* In Proceedings of the IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), pp. 185-191. IEEE Computer Society, 2012.

[17] I. Y. Jung, B. J. Han, C. S. Jeong. *Provisioning On-Demand HLA/RTI Simulation Environment on Cloud for Distributed-Parallel Computer Simulations.* Journal of the Mobile, Ubiquitous, and Intelligent Computing, 274: 329-334, 2014.

[18] X. Chai, Z. Zhang, T. Li, Y. Zhang, B. Hou. *High-performance cloud simulation platform advanced research of cloud simulation platform.* In proc. of the Grand Challenges on Modeling and Simulation Conference, pp. 181-186, 2011.

[19] B. H. Li, X. Chai, B. Hou, C. Yang, T. Li, T. Lin, Z. Zhang, Y. Zhang, W. Zhu, Z. Zhao. *Research and application on cloud simulation.* In proc. of the Summer Computer Simulation Conference, pp. 34: 1-14, 2013.

[20] S. Feng, Y. Di, Z. X. Meng. *Remodeling traditional rti software to be with paas architecture.* In proc. of the Computer Science and Information Technology, 1: 511-515, 2010.

[21] *poRTIco*. 2009. [Online]. Available: http://www.porticoproject.org/. Accessed on January 6, 2015.

[22] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield. *Xen and the art of virtualization.* In proc. of the ACM Operating Systems Review, 37(5): 164-177, 2003.