OBJECT-BASED VIEWPOINT FOR LARGE-SCALE DISTRIBUTED VIRTUAL ENVIRONMENT

Elfizar[1], Mohd Sapiyan Baba[2], and Tutut Herawan[3]

[1,3]Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia
[1]Department of Information System, University of Riau, Pekanbaru 28293, Indonesia
[2]Gulf University of Science and Technology, Kuwait City, Kuwait

Email: [1]elfizarmd@gmail.com, [2]sapiyan.m@gust.edu.kw, [3]tutut@um.edu.my

ABSTRACT

Distributed Virtual Environment (DVE) is a shared application consisting many objects, which can be accessed by many users. There have been many methods used to scale the DVE such as dividing simulation workload, dynamic load balancing among servers, and creating alternative architectures. However, they may not accommodate many objects and users. In this paper, we explore all approaches used to scale the DVE and then determine the characteristics of the existing approaches. With those characteristics, we compared existing approaches based on three parameters: the number of simulation per region, implementation, and the number of objects managed by simulator. The results show that all approaches use the same viewpoint, called present viewpoint, in developing the DVE. It views DVE as a world where all objects and activities are managed by a simulator. The results also show that this viewpoint contributes in terms of limitations of the current DVEs performance. In response to these results, we further propose a new viewpoint, called object-based viewpoint, to generate object-based simulators architecture. The experiment results show that our proposed architecture can provide a large scale DVE with better performances than the previous architectures.

Keywords: Distributed virtual environment, DVE scalability, Object-based viewpoint, Object-based simulators

1.0 INTRODUCTION

Virtual Environment (VE) as a simulation application is widely studied and used for the development of computer generated synthetic environments and analysis purposes. To involve many users in a VE, Distributed Virtual Environment (DVE) is often needed. Many users in separated places can come together to collaborate in a VE. For instance, they can use DVE to collaborate virtually with each other to carry out a work such as surgery training, automotive assembly simulations, etc. They can also go to a virtual music concert or attend a virtual classroom.

Virtual world is one of the most popular applications of DVE. For instance is Second Life [1], which is the state of the art of virtual worlds. On May 2012, the world of Second Life was made up of thousands of regions, which if they are linked together will spread over 1,962.93 km^2 of virtual lands [2]. The world consists of avatars, terrains, trees, buildings, and other objects. Each region is a process run by a simulator. With Second Life, users can enjoy the 3D scenery, walk, drive, interact with other avatars, play games, or create objects. In fact, 99% of objects in Second Life are user created [3]. This virtual world is often used commercially by users to sell their properties to others.

Therefore, DVEs may have a very large number of objects and users at one time and this can easily overload a fast network, as well as impose huge processing requirements at the server and client computers. As computing resources are limited, there are obvious problems that arise once the number of objects and users in a simulation reach a certain limit. If no special mechanisms are provided, one may expect a DVE to produce undesirable effects such as choppy rendering, and loss of interactivity, due to lack of processing power to handle the increasing load.

Scaling a DVE depends on two aspects, i.e. scaling the number of concurrent users interacting with each other, or scaling the scene complexity (number of objects and the complexity of their behaviours and appearances). Several methods have been generated to scale DVEs such as dividing simulation workload [4-6], using dynamic load balancing among servers [7], and creating alternative architectures [8-11]. Scaling the DVEs can be done at the server's side (using cluster or cloud computing) or the client's side (using peer-to-peer model). However,

these techniques are not enough to accommodate DVEs with huge number of objects and thousands of concurrent users. Other than that, increasing the number of objects and users decreases the performance of DVE.

In this paper, we determine the characteristics of the present DVEs. These characteristics are then used to determine the present viewpoint to develop the DVEs. To address the limitation of current approaches, we propose a novel viewpoint, called object-based viewpoint, to generate a new DVE architecture. The novelty of the proposed approach is that, unlike the existing viewpoint approaches, the proposed object-based viewpoint views the DVE as a world that consists of many objects and each of them is able to manage itself for the appearances and behaviours as well as interaction with other objects.

In summary, the contributions of this paper are described as follows:

a. We study characteristics of the current approaches used to develop DVE. With these characteristics, we are able to find the present viewpoint in scaling the DVE.
b. We introduce object-based viewpoint, a new viewpoint to generate a large scale distributed virtual environment architecture, called object-based simulators architecture.
c. We do experiments and prove that our proposed architecture can provide a large scale DVE with better performances than the previous architectures.

The rest of the paper is organised as follows. Section 2 presents the current approaches used by researches in scaling the DVE. The characteristics of the current approaches are also described in this section. Section 3 presents the novel viewpoint to scale the DVE. The experiment results using novel DVE architecture are described in Section 4. Finally, Section 5 and Section 6 give some conclusive remarks and future work, respectively.

2.0 RELATED WORKS

In current DVEs, entities and activities are managed by a simulator. When the simulator workload increases, the current approaches use two techniques to scale the DVE i.e. splitting the region and separating the component of DVE simulator. In the following section, we describe these techniques.

2.1 Splitting the Region

To decrease the simulation workload, this approach divides the region into smaller areas and each area is simulated by a simulator. Thus, the simulation workload of each area becomes smaller.

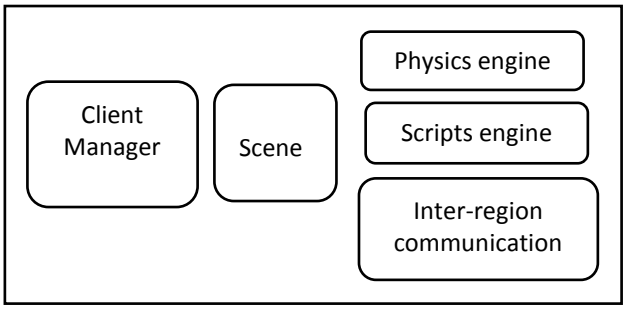

Fig. 1: Simulator components of simulator-centric architecture

Second Life as a famous DVE application uses this approach. The world is divided into a large number of small regions. Each region has area of 256m x 256m and owned by exactly one simulation server. OpenSim [12], an open source DVE that has a system's architecture similar to and compatible to Second Life architecture, also uses this approach. Both Second Life and OpenSim have simulator-centric architecture, and simulator for each region is illustrated by Fig. 1. In this figure, all simulator components i.e. scene, physics engine, client manager, etc run in a process that has responsibility to manage all entities and activities in a region.

Yamamoto *et al*. [13] used splitting the region approach to split the region. They divided the game world into identically sized regions for management of resources and propagation of messages. Lee [14] divided the DVE into even sized regions determined by the number of servers processing data, e.g. 16 grids per server, with 16 servers, for a total of 256 regions. Jardin and Zappala [15] proposed a hybrid architecture using fixed regions to limit propagation of avatar movement messages. De Vleeschauwer *et al*. [16], Ahmed and Shirmohammadi [17] exploited the concept of microcells, which have a relatively small size with respect to the entire VE dimension.

Most current Massively Multiplayer Online Games (MMOGs) also used this static region-partitioning model. To prevent server crashes, game operators have resolved to use *sharding* [18]. They make replicas of particularly popular region. The World of Warcraft [19], the most popular MMOG also used this *sharding* method.

Sometimes, regions are dynamically rather than statically created. Liu and Bowman [7] used binary space partitioning (BSP) to partition the world in order to scale the DVE flexibly by dynamic allocation of hardware to match load. Although this method was effective in balancing workload dynamically, it had several limitations and suffered from high overhead of workload migration. A similar approach was also proposed by introducing dynamic and fair distribution of load based on physical partitioning of a virtual space [5]. Whenever a server exceeds its maximum capacity, a new server was added and the region load was shared with it.

To increase flexibility of resource allocation and address the over-provision problem when the peak load occurs, managing each DVE region can be integrated with cloud computing [20-23]. In a DVE cloud environment, virtual machine (VM) is allocated to serve clients of a region, instead of DVE server. Cloud computing follows the utility computing model based on a resource pay-per-rent model allowing customers to pay only the resources they actually use. Game operators may exploit clouds by requesting a large set of resources during peak hour and by releasing them when they are no longer needed.

Peer-to-peer (P2P) is a new solution to scale up the DVE. Generally, it distributes the server roles to peers. Chan *et al*. [24] and Olanda *et al*. [25] proposed hybrid approaches where the world was divided into regions and each region was assigned to a peer belonging to a structured P2P overlay. Several approaches allow only those peers that satisfy particular requirements in term of hardware capability to manage part of the simulation. This special peer is referred as Super Peer (SP) and usually manages a region so that each peer in the region is connected to it. Chen and Muntz [26] used a SP to manage each hexagonal region while Kim *et al*. [27] and Buyukkaya *et al* [28] used a SP to manage a square and polygon region, respectively.

Splitting the region can also be done by considering the Area of Interest (AOI). Several works such as Almashor *et al*. [29], and Carlini *et al*. [30] use Voronoi overlays to maintain AOI, which eases the identification of neighbours. In a Voronoi overlay network, each peer manages the space correspondent to its Voronoi region. Denault *et al*. [31], and Van Den Bossche *et al*. [32] treated regions as set of adjacent microcells. To improve the DVE performance, Ranjan and Zhao [33] and Carlini *et al*. [6] used the hybrid of P2P and Cloud architecture.

2.2 Separating the Components

It is the second approach used by researches to scale up the DVE. Even though the region has been split into sub-regions, the problem still occurred as the number of objects or users in a sub-region increase dramatically. To address the problem, the researches generate alternative architectures. Some simulator components are separated from the main simulator so that the workload of main simulator can decrease. The separated component does its tasks in independent process and then the results are sent to main simulator.

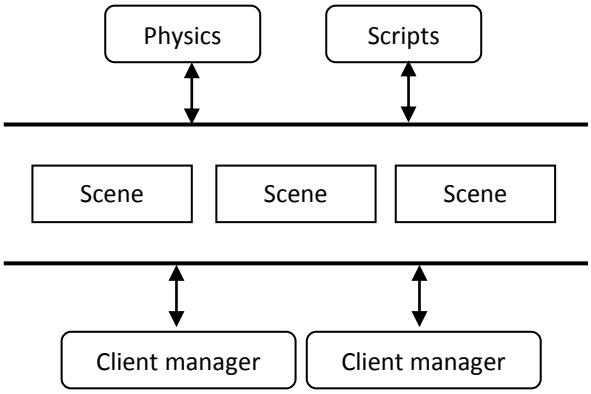

Fig. 2: DSG architecture

Distributed Scene Graph (DSG) [10] uses this approach. It views the DVE operations in general as a collection of the "Scene" and the actors operating on the Scene through a Scene service layer (Fig. 2). Actually, DSG is inspired by Dakstar [8] that implements DVE logic as small transactional tasks distributed across servers.

In Fig. 2, DSG separates the physics and script engine as well as client managers from the main simulator. Each actor is responsible to do its task when the simulation runs, and all actors are mediated by Scene. Each actor observes a part of the virtual world and executes operations to the objects in this part. The DSG architecture allows developer dynamically to distribute the load depending on the current situation in the world. For example, if at one place there are a lot of events, which involve many users, then it makes sense to dynamically allocate one server to perform user management tasks for that particular place and maybe just one more server to handle users in all other parts of the world. This allows one to efficiently reuse the hardware and thus decrease the cost of running a virtual world [9].

Another architecture is Sirikata [11, 34]. Sirikata's architecture splits the functionality of the platform into three concepts i.e. space management, object simulation, and content delivery, as shown by Fig. 3. It is different from the traditional approach where all objects together with their scripts and data are simulated on a single server or a cluster.

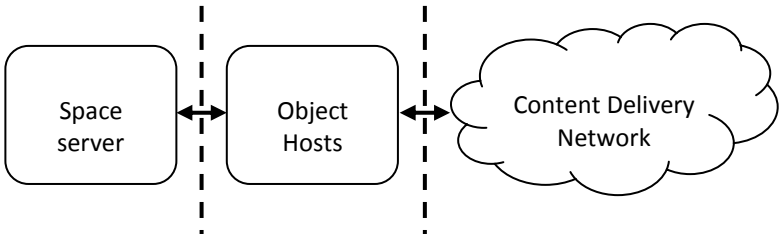

Fig. 3: Sirikata architecture

2.3 Characteristics of the Current Approaches

Table 1 illustrates the characteristics of splitting the region and separating the component approach. In this table, we measure both approaches with three parameters i.e. the number of simulator per region, implementation, and the number of objects managed by simulator:

- Number of simulator per region
  It represents the number of simulator used to manage a region in the DVE.
- Implementation
  It describes the network used to run the simulation.
- Number of objects managed by simulator
  It represents the number of objects handled by a simulator. For the second approach (separating the component approach), the value is measured for each simulator.

Table 1: Characteristics of scalability approaches

Parameters	**Scalability Approach**	
	Splitting the Region	**Separating the Component**
Number of simulator per region	One	Many
Implementation	CS, P2P	CS, P2P
Number of objects managed by simulator	Many	Many

From Table 1, we note that all methods in both approaches can run on client server (CS) or peer-to-peer (P2P) network. The difference of both methods only occurs on the first parameter where the first approach has one simulator for a region whereas the second one has many simulators in managing the region. Increasing the number of simulator in the second approach is influenced by separating the simulator components from the main simulator. Even though both approaches have different number of simulator per region, but they have similarity in number of objects managed by each simulator i.e. more than one object.

We carried out an experiment that compared the CPU usage and memory allocation between splitting the region (SR) and separating the component (SC) approach for varying number of objects in DVE. The experiment was intended to measure the influence of the number of simulator in DVE with respect to the DVE performance. In SR, we used a simulator that managed many objects in a region. The simulator handled all objects appearances and behaviours. SC separated the physics engine from the simulator. Thus, there were two simulators: physics simulator and the main simulator handling all activities of the world except physics simulation. As shown by Fig. 4, the experiment result shows that the performance of the second approach (SC) is better than SR. The CPU usage of SC just tends to increase linearly while SR tends to increase quadratic.

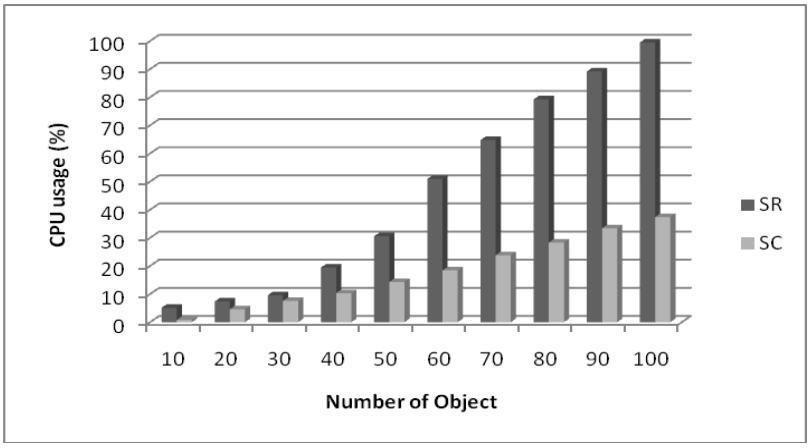

Fig. 4: CPU usage of both approaches

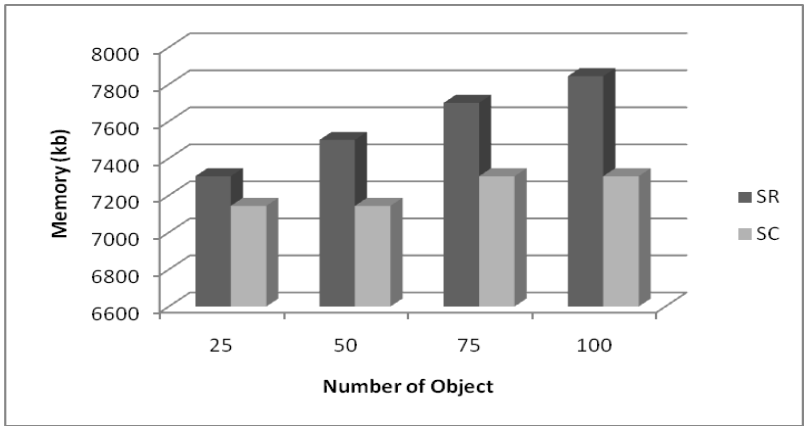

Fig. 5: Memory allocation for both approaches

The better performance of SC also occurs in the memory allocation for the simulator. As depicted by Fig. 5, the SC simulator uses less memory than SR for varying number of objects. The increasing number of object does not increase linearly the memory allocation of SC as occurred to SR. Based on both results, we note that using more simulators yields better DVE performance than using one simulator. In other words, separating the components approach is able to improve the performance of the splitting the region approach in order to scale up the DVE.

3.0 PROPOSED OBJECT-BASED VIEWPOINT METHOD

Splitting the region approach used to scale up the DVE still has many limitations. This approach makes user isolated from others. A user is only able to live in a small area of environment. If there are many objects and users in this area then the DVE performance degrades. Even though these limitations could be addressed by separating the component approach as discussed in the last section, the limitations still occur when the number of objects and users residing in an area increases. It will increase the workload of actors or object host in DSG or Sirikata, respectively. It means that addressing the DVE scalability problems is not sufficient with modifying the last architecture and then repairing its functionality or component. Hence, the viewpoint used by researches to develop current architectures should be further investigated to discover the core of the problem instead of its symptoms.

Based on the characteristics of the current approaches as illustrated by Table 1, it is evident that all researches have similar viewpoint in developing the DVE architecture. They view DVE as a world that all activities and components are handled by a "something" (called simulator). When this simulator approaches its capacity limit in managing the world, the world has to be split into smaller sub-world and each of them is managed by instance of simulator. Another approach is separating the component of simulator to be an independent component in processing its task.

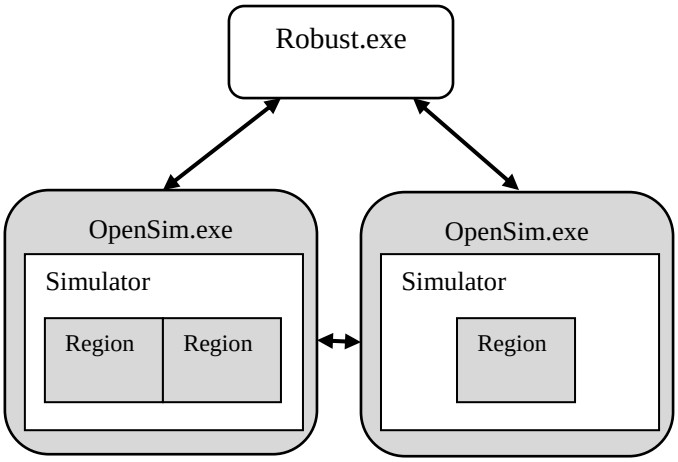

Fig. 6: OpenSim architecture

Fig. 6 shows the architecture of OpenSim. This architecture uses that viewpoint. All objects and users activities are handled by a simulator or process (OpenSim.exe). To improve the OpenSim performance, splitting the region approach drives the developers to divide the region, and separating the component approach separates the simulator components from the main simulator. Unfortunately, when workload in a region or scene increases dramatically, the workload of simulator also increases automatically. It is the main problem why increasing DVE complexity also decreases the performance of the DVE.

To solve the problem, we should change the current viewpoint above. DVE does not consist of regions that need to be split into smaller areas, and it does not require separating one or more components from the main simulator when the simulator workload increases.

3.1 Object-based Viewpoint

To change the current architectures in providing large scale DVE, a new viewpoint has to be established. In the proposed viewpoint, we look at DVE as a collection of objects. Trees, animals, houses, boxes, avatars are examples of these objects. Several objects are static in the environment, and some of them are dynamic objects that are able to interact with other objects. Those objects compose the DVE and each is an independent process in the VE that can determine its appearances and behaviours in the environment. For example, an avatar is an object. Thus, avatar is a process in VE that has specific appearances and behaviours. How the avatar should walk, handshake, sit, and other activities are maintained by the process. The avatar may reside in any region or scene in virtual environment using those appearances and behaviours. The region just uses this process. This new viewpoint is called object-based viewpoint.

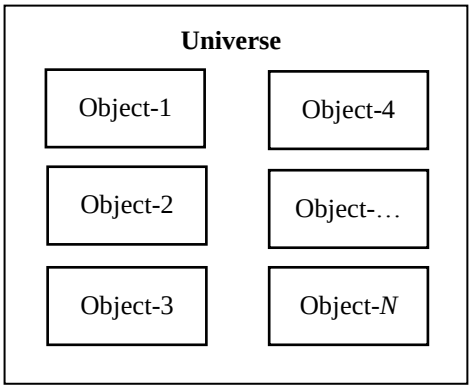

Fig. 7: DVE structure

The object-based viewpoint is inspired by the real world environment. If we look at around, we can easily see person, trees, grass, buildings, animals etc. All of them are objects that compose the world and each has distinct appearances and behaviours controlled by the object itself. With the object-based viewpoint, the DVE can be illustrated by Fig. 7. It consists of several objects residing in universe and each is independent process and separated from other objects. In this paper, we use term of *universe* to show that objects reside in a continuous space, not a partitioned space as used by the current DVE architectures. A process handling the appearances and behaviours of an object is called object simulator or simulator for that object.

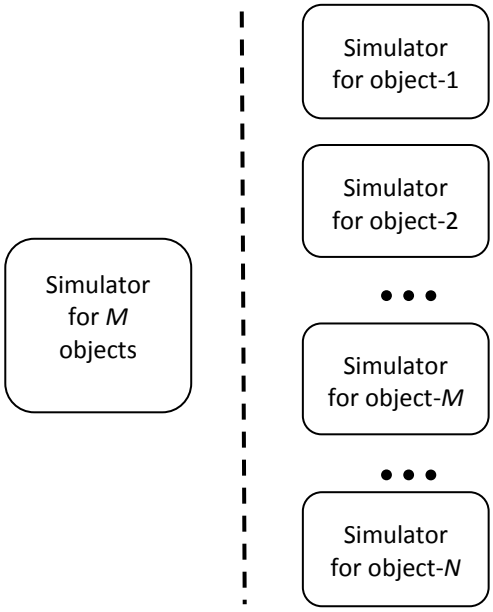

Fig. 8: Simulator of both viewpoints

Fig. 8 shows the difference between simulator used by the current and object-based viewpoints. The left side of this figure represents the current viewpoint where a simulator simulates many objects. In this case, we assume a simulator simulates M objects where $M \leq N$. We use this assumption because a current DVE may consist of more than one simulator e.g. Second Life and OpenSim. Further, the right side of this figure shows the simulators which are based on the object-based viewpoint. It is new viewpoint to create a large-scale DVE architecture. A simulator simulates just one object residing in the VE. From Fig. 8, some advantages of object-based viewpoint are as follows:

- Object simulator is an independent process so that it can reside in distributed computers. Thus, it is scalable with the additional hardware.
- Each object can be identified by its simulator ID rather than server ID. Therefore, the migration of simulator does not affect the overall simulation process.
- The workload of object simulator is less than current DVE simulators. The increasing number of objects in DVE does not affect the simulator itself. It eliminates the workload distribution algorithm among servers as used by the current DVE.

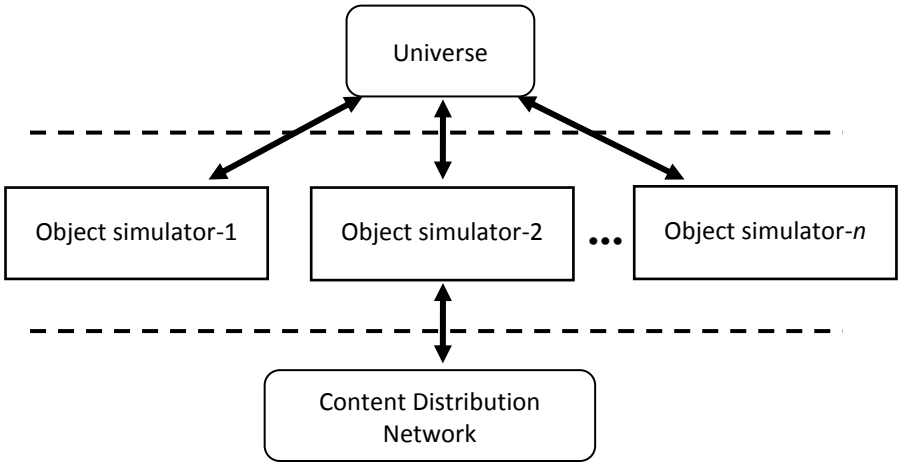

Fig. 9: 1P1O architecture

3.2 Object-based Simulators Architecture

With object-based viewpoint in Section 3.1, in the proposed architecture, simulator is no longer a single process that controls many objects. Each object is treated as a process or simulator. Hence, this architecture is called 1P1O model: one process for one object. This model makes object independent in managing itself. The 1P1O model has three components, i.e. object simulators, universe and content distribution network (CDN) as shown in Fig. 9.

3.2.1 Object Simulator Component

The core component of 1P1O model is simulators that comprise several object simulators. Object simulator is responsible to simulate the appearances and behaviours of the object. Fig. 9 shows that there are *n* object simulators simulating *n* objects in VE. The object simulator has two components used in managing an object:

- Script engine
  It is used to run the object scripts that determine the appearances and behaviours of object.
- Physics engine
  Physic engine runs the physics simulation of the object. It ensures that the object enforces physics laws. Examples are gravity pole, collision handling, etc.

An object simulator is a process in DVE, allowing it to specify the behaviour and appearance of object and providing it access to the universe component. The viewer is an object in DVE so that it is also represented by a process. Different from other objects, the viewer is able to display the environment. A user in VE may be represented by avatar. One important thing is that each simulator can reside in different hardware since it is an independent process. Thus, it is scalable with the additional hardware.

3.2.2 Universe Component

The universe component in 1P1O model is responsible to determine what objects are in the environment. This component has similarity to the scene in DSG architecture and the space server in Sirikata architecture. It stores object properties in a VE such as object identifier, position, and physical properties. A given universe may be run by one or more computers (called universe server) which segment the geometric coordinates of the VE. Similar to object simulator, the universe component is scalable to the additional hardware. Since universe component stores the location and properties of objects, it is able to synchronize the objects position and properties among interested simulators. All object simulators send updates to universe, and universe disseminates those updates to interested simulators. Universe is also responsible to make an object know the nearest objects.

3.2.3 Content Distribution Network

Content Distribution Network (CDN) is similar to CDN component in the Sirikata architecture. It stores permanent data and delivers it to the other components. Meshes that are used to display objects are examples of

the data stored on CDN. Viewers are able to download them to view the world. The CDN may be as simple as a web server. It really needs only to serve requests for files.

4.0 RESULT AND DISCUSSION

4.1 Experimental Setup

Fig. 10 shows the 1P1O model that is implemented in CS (client-server) network. All object simulators reside in server(s). The universe is run on one or more servers, and so is CDN. Clients are viewers that view the DVE and may interact with other objects. When a user moves in environment, object appearances and behaviours are managed by its simulator residing in a certain server, and the state updates are sent to universe. On a certain case, where the server capacity exceeds its limit, the migration of simulators to another server are done easily because the reference of an object is based on the object ID, not based on server ID.

The objects used by these applications are boxes with varying sizes and masses. The application simulates the objects falling from a certain high to the ground and colliding with each other. The collision occurs between object and ground or among objects. Fig. 11 shows the falling objects in the VE, and they can be viewed from different viewpoint. In this figure, each object appearances and behaviours are simulated by a simulator.

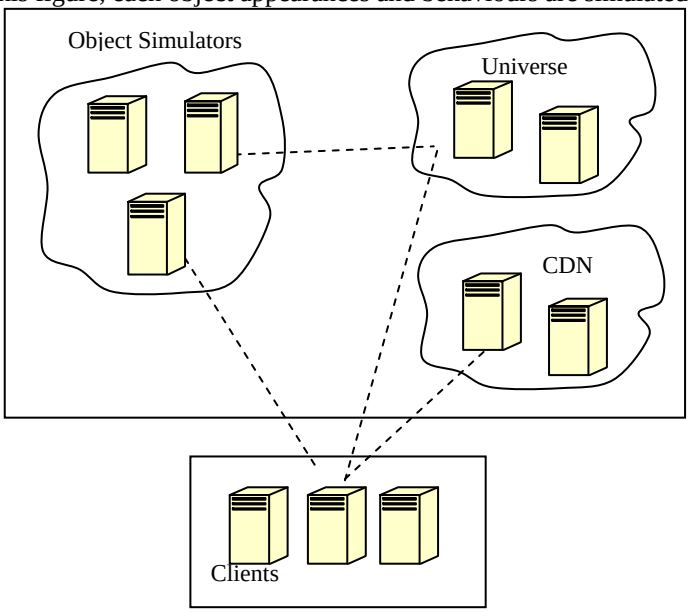

Fig. 10: 1P1O implementation

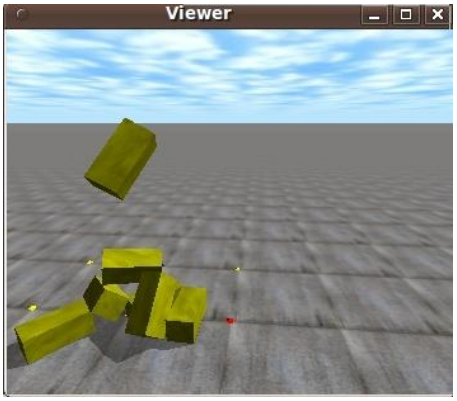

Fig. 11: Falling Objects

The objective of this experiment is to evaluate the 1P1O model in the CS network. Two parameters used in the evaluation process are as follows:

- Scalability of the model

It is measured by using the frame rates of simulation with respect to varying number of objects and users involved in the DVE.

- Performance of the model
  It is measured by the CPU usage and memory allocation for simulator.

The simulators are run on a computer with dual core 1.6 GHz processor. It acts as computer server. The viewers are executed on a computer with the same specification as the computer server. Further, the universe is run on computer with Intel Atom 1.66 GHz processor. Both the 1P1O (proposed architecture) and Sirikata model (the current architecture) are executed on this platform. This experiment uses varying number of objects and concurrent users.

Some variables measured in this experiment are as follows:

- The frame rates of universe and space with respect to the number of objects and users
- CPU usage of simulator for both models
- CPU usage of universe and space
- Memory allocation for simulator of both models
- Memory allocation for universe and space

The first variable is used to determine the scalability of the model. We use this variable to know the effect of increasing number of objects and users with respect to the frame rate of simulation. The other variables are used to determine the performance of our model compared to the current model.

4.2 Experiment Results

The experiment results presented and discussed in this section are based on the output of the experiment. The subsections below are described based on the variables measured in the experiment.

Table 2: Universe frame rates based on objects

Number of objects	Universe Frame Rates (fps)						
	1	2	3	4	5	6	Average
10	59.038	59.753	58.923	59.364	59.108	59.232	59.236
20	59.155	59.318	59.363	58.913	59.050	59.353	59.192
30	59.142	59.123	59.554	58.722	59.046	59.558	59.191
40	59.165	58.315	58.959	58.921	58.746	58.932	58.840
50	59.123	59.164	58.913	59.644	59.159	58.120	59.021
60	59.134	58.240	58.946	58.937	58.641	58.639	58.756
70	58.321	59.037	58.747	58.736	58.618	59.360	58.803
80	58.523	58.631	59.182	58.704	58.765	58.510	58.719
90	58.316	58.467	58.339	59.111	58.954	57.942	58.522
100	58.775	57.924	58.560	58.746	58.571	58.380	58.493

4.2.1 Frame Rates of Model Based on Objects

The frame rates of the universe for varying number of objects are shown in Table 2. These frame rates are measured from the condition where universe has already received all updates and disseminated the updates to interested objects to thirty seconds after that condition. Hence, there are six measurements because the frame rate is displayed in every five seconds. The frame rates are stated in frame per second (fps), and the average frame rates is provided at the right side of table.

Generally, the frame rates of universe depend on the number of objects residing in the VE. It decreases with the increasing number of objects. The increasing of average frame rates in row 5 and 7 indicates that there are a few objects collisions occurred for 50 and 70 objects, respectively.

Table 3 illustrates the frame rates of space in Sirikata model. These frame rates are measured by using the same condition as 1P1O model. Thus, they also consist of six measurements for varying number of objects. This figure also shows that the frame rates of simulation decrease as the number of objects increases. The comparison between average frame rates of Sirikata and 1P1O model is illustrated by Fig. 12. The difference between both

models is striking where the frame rates of Sirikata are lower than 1P1O model. Another important fact is that the 1P1O frame rates do not persistently decrease as Sirikata.

Table 3: Space frame rates based on objects

Number of objects	Space Frame Rates (fps)						
	1	2	3	4	5	6	Average
10	59.378	58.613	58.472	58.528	59.363	59.187	58.924
20	59.152	59.330	58.965	58.749	58.207	59.047	58.908
30	59.251	57.769	58.340	58.384	58.186	58.518	58.408
40	56.988	58.077	57.598	58.952	57.993	57.787	57.899
50	56.551	57.767	56.330	57.154	56.823	57.549	57.029
60	57.092	56.890	56.956	56.881	57.192	56.166	56.863
70	56.767	57.362	57.327	56.712	56.322	56.767	56.876
80	57.533	56.122	55.833	55.512	56.598	56.165	56.294
90	56.516	56.255	56.578	55.580	55.783	56.966	56.280
100	55.321	55.913	55.780	55.986	56.184	55.477	55.777

4.2.2 Frame Rates of Model Based on Users

Users view the VE using the viewer application. As described before, viewer is an object that is able to display the VE. Table 4 illustrates the frame rate of 1P1O model for varying number of users. These values are also measured from all tasks already done by universe component to thirty seconds after that. Similar to Table 2, the frame rates of the 1P1O model decrease as number of viewers (users) increases. Increasing values of row 4 and row 8 are caused by many collisions occurred among user's representation compared with the collisions for row 3 and row 7, respectively.

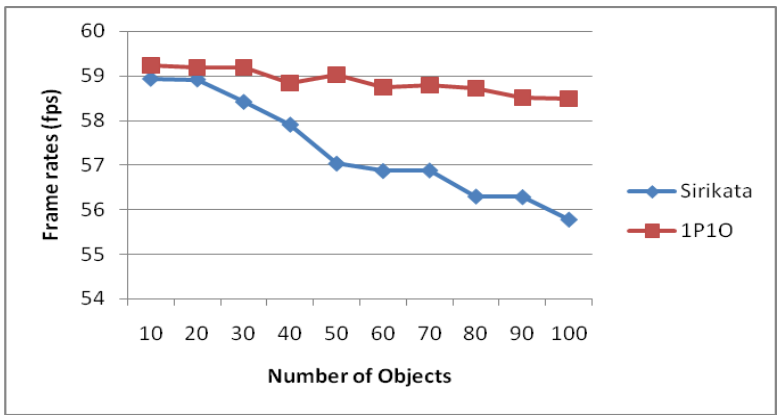

Fig. 12: Average frame rates of models based on objects

Furthermore, the frame rates of Sirikata are drawn by Table 5. They are measured from frame rates of space, which determines other components in Sirikata architecture. In fact, similar to 1P1O, the Sirikata frame rate also decreases as number of users in VE increases. It means that increasing number of users increases the workload of simulation. Comparison of average frame rates between 1P1O and Sirikata model can be seen in Fig. 13. From the figure, we found that the frame rate of 1P1O is higher than Sirikata. The increasing number of users involved in VE does not strictly decrease the frame rate of 1P1O. Hence, the 1P1O model is more scalable with the increasing number of users in VE.

Table 4: Universe frame rates based on users

Number of viewers	Universe Frame Rates (fps)						
	1	2	3	4	5	6	Average
10	59.356	59.562	59.594	59.775	59.560	59.127	59.496
20	59.395	59.761	59.358	59.390	59.676	58.958	59.423
30	58.936	58.915	59.362	59.517	59.548	59.121	59.233
40	59.320	59.146	59.323	59.439	59.445	59.547	59.370
50	59.167	58.989	59.380	59.310	59.250	59.149	59.208
60	59.367	59.344	59.333	58.703	59.045	59.166	59.160

70	59.373	59.434	59.569	58.513	59.359	58.322	59.095
80	59.352	58.760	59.569	58.717	59.558	58.924	59.147
90	57.960	59.315	59.110	59.338	59.374	58.755	58.975
100	59.748	59.354	59.327	59.120	58.734	57.354	58.940

Table 5: Space frame rates based on users

Number of viewers	Space Frame Rates (fps)						
	1	2	3	4	5	6	Average
10	59.189	59.240	59.248	59.230	59.566	58.519	59.165
20	58.709	59.095	58.994	59.664	58.395	59.041	58.983
30	58.161	58.766	57.554	57.530	58.715	58.074	58.133
40	57.902	57.912	57.113	58.169	57.551	57.647	57.716
50	58.183	57.829	57.530	58.534	58.351	58.185	58.102
60	57.993	57.704	57.167	58.126	57.107	56.561	57.443
70	56.198	57.690	56.768	57.185	58.111	57.570	57.254
80	56.117	56.918	57.199	57.200	57.827	58.461	57.287
90	55.981	56.364	56.338	57.521	57.793	57.323	56.887
100	56.971	56.372	55.913	56.182	56.567	57.033	56.506

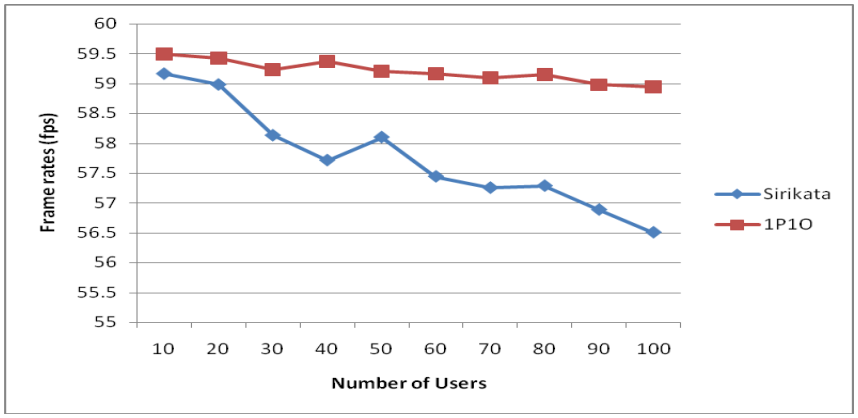

Fig. 13: Average frame rates of models based on users

4.2.3 CPU Usage of Simulator

The CPU usage of both 1P1O object simulator and Sirikata object host for varying number of objects are illustrated by Fig. 14. These values are taken from simulator running in the environment. From the figure, the Sirikata uses CPU time longer than 1P1O model. One of the reasons is that Sirikata object host simulates many objects in simulator so that its workload depends on the number of those objects.

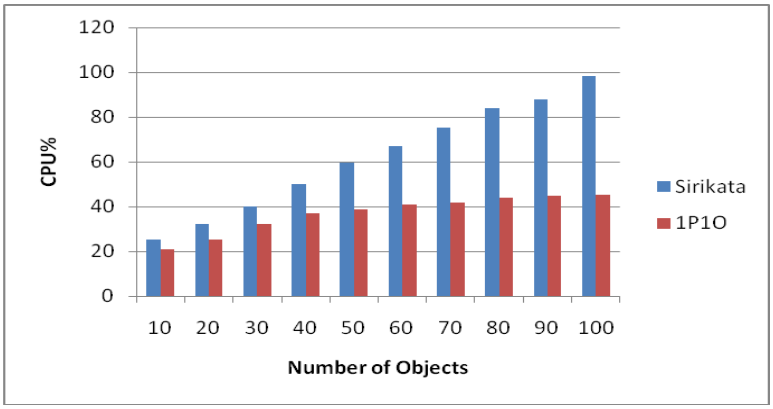

Fig. 14: Simulator‘s CPU usage

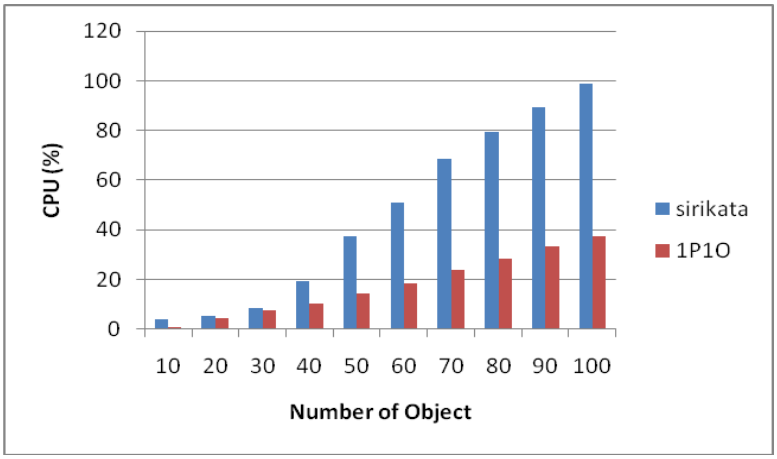

Fig. 15: Simulation CPU usage

4.2.4 CPU Usage of Model

The CPU usage of universe and space for both models for varying number of objects are depicted by Fig. 15. The Sirikata model has CPU time greater than 1P1O model, and the disparity between both models is very high. For example, the CPU time of Sirikata for 50 objects is almost similar to CPU time of 1P1O model for 100 objects. This behaviour is the same as CPU usage of simulator. This means that CPU usage of simulator influences the CPU usage of model.

4.2.5 Memory Allocation for Simulator

Memory allocation is determined by using the virtual memory used by a simulator. The memory allocations for simulator of both 1P1O and Sirikata model are shown by Fig. 16. We note that Sirikata simulator uses memory larger than 1P1O object simulator. The linear trend-line of Sirikata is not followed by the 1P1O in the figure. The 1P1O simulator has constant memory allocation for a certain interval of number of objects.

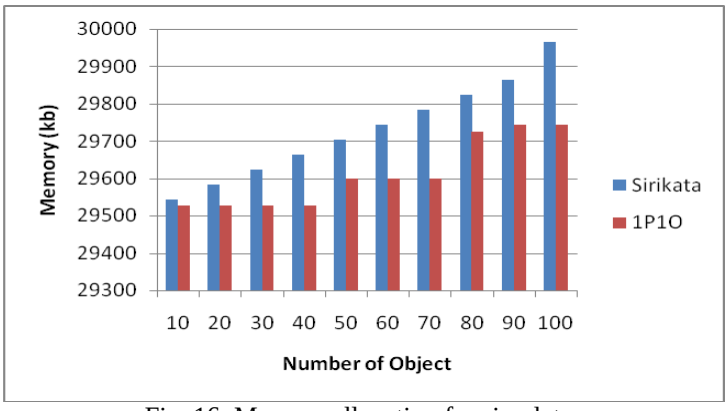

Fig. 16: Memory allocation for simulator

4.2.6 Memory Allocation for Model

Similar to CPU usage of model, the memory allocation for 1P1O and Sirikata model is measured from universe and space component, respectively. The experiment result is illustrated by Fig. 17. From the figure, we note that although the memory allocation for Sirikata fluctuates, its values are higher than the 1P1O model.

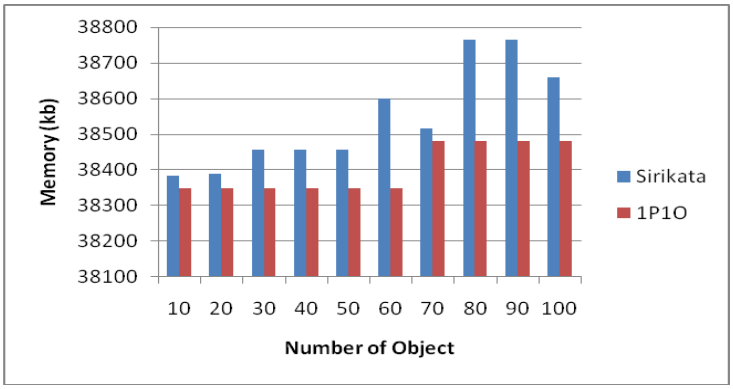

Fig. 17: Memory allocation for model

4.3 Discussion

The experiment results show that frame rates of 1P1O model are higher than Sirikata for both varying number of objects and users. As illustrated by Fig. 12 and Fig. 13, the frame rates of the current model persistently decrease as the number of objects and users increase. The workload of simulator also increases significantly when the number of object and users increase in the DVE. It causes the frame rates of the current model decrease. Fortunately, it does not happen to the 1P1O model since a distinct simulator manages each object. The increasing number of objects or users does not decrease the frame rates of model significantly. Thus, 1P1O model is able to accommodate the increasing number of objects and users in the VE. It means that the scalability of 1P1O model is better than the current DVE model. It is able to provide better user experiences than the current model.

Furthermore, the CPU usage of the 1P1O model is less than the current model in both the simulator and universe. In Fig. 14, the CPU usage of the current model's simulator is linear with respect to the number of object. The increasing number of objects increases the CPU usage of simulator linearly. Contrary to Sirikata, our model simulator has CPU usage trend-line of logarithmic. It is better than linear CPU usage. Fig. 15 also shows that 1P1O model has CPU usage less than the current model. Our model has linear trend-line, but unfortunately, the CPU usage of the current model increases quadratic with respect to the increasing number of objects.

The good result of our model is also followed by the memory allocation for simulator and model. The memory allocation of 1P1O is always less than the current model. It is depicted by Fig. 16 and Fig. 17. There are constant values of memory allocation in a certain interval in our model since the simulator only simulates the same number of objects. Unfortunately, the current model has different memory allocation values with respect to increasing number of objects and these values are always higher than 1P1O model. With two measurements above (CPU usage and memory allocation), the 1P1O model has better performance than the current model.

5.0 CONCLUSIONS

Currently, the scalability problems still challenge the researches in the DVE. To address the problems, this paper investigates the present methods used by researches to scale the DVE. All of these methods are classified into two approaches i.e. splitting the region and separating the component of simulator. The experiment results show that the second approach has better performance than the first one in both CPU usage and memory allocation for simulator.

We have determined the characteristics of both approaches with three parameters i.e. the number of simulator per region, implementation, and the number of objects managed by simulator to find out the main problem. Based on these characteristics, we found some limitations of these approaches which are caused by the viewpoint used by developers to create and modify the DVE architecture. All approaches view the DVE as a world where all activities and appearances are managed by a simulator. It is called present viewpoint, and it is another finding of our work.

The present viewpoint influences the scalability of the DVE. Increasing the number of objects and users reduces the DVE performance. Hence, the viewpoint change is required. This paper proposed a novel viewpoint called object-based viewpoint. It views the DVE as a world that consists of several objects. Each object is independent from each other. All behaviours and activities of the object are simulated by its own process or simulator.

The object-based viewpoint is implemented in developing a large scale DVE, called object-based simulator architecture or 1P1O model. The experiment results show that our model is more scalable with better performance than the current model. This model can be used by DVE developers and other researches to accommodate huge number of objects and users in the DVE.

6.0 FUTURE WORK

We have proposed a novel model that is able to accommodate a large scale DVE. In the experiments, we have implemented the model in client server network. Our future work is to implement this novel architecture in peer-to-peer (P2P) network. Based on the advantages of the P2P network, it is possible to get more scalable DVE.

ACKNOWLEDGEMENT

The work of Tutut Herawan is supported by PRPUM Research Grant No CG063-2013. The work of Elfizar is supported partly by University of Riau. The helpful comments and suggestions of the reviewers are acknowledged.

REFERENCES

[1] Second Life, 2014, http://www.secondlife.com.

[2] T. Shepherd, "Second Life Grid Survey – Region Database", 2012, http://www.gridsurvey.com.

[3] C.R. Ondrejka, "Escaping the Gilded Cage: User Created Content and Building the Metaverse", *New York Law School law review*, Vol. 49, No. 1, 2004, pp. 81-101.

[4] W. Cai, P. Xavier, S.J. Turner, B. Lee, "A Scalable Architecture for Supporting Interactive Games on the Internet", *In Proceedings of the Sixteenth Workshop on Parallel and Distributed Simulation*, 2002, pp. 60-67.

[5] U. Farooq, J. Glauert, "Scalable and Consistent Virtual Worlds: An Extension to the Architecture of OpenSimulator", *In Proceedings of Int. Conf. on Computer Networks and Information Technology*, 2011, pp. 29-34.

[6] E. Carlini, L. Ricci, M. Coppola, "Flexible Load Distribution for Hybrid Distributed Virtual Environments", *Future Generation Computer Systems*, Vol. 29, No. 6, 2013, pp. 1561-1572.

[7] H. Liu, M. Bowman, "Scale Virtual Worlds through Dynamic Load Balancing", *In Proceedings of IEEE/ACM Symposium on Distributed Simulation and Real Time Applications*, 2010, pp. 43-52.

[8] J. Waldo, "Scaling in Games and Virtual Worlds", *Communications of ACM*, Vol. 51, No. 8, 2008, pp. 38-44.

[9] S. Byelozyorov, R. Jochem, V. Pegoraro, P. Slusallek, "From Real Cities to Virtual Worlds using an Open Modular Architecture", *The Visual Computer*, Vol. 28, No. 1, 2012, pp. 1-13.

[10] D. Lake, M. Bowman, H. Liu, "Distributed Scene Graph to Enable Thousands of Interacting Users in a Virtual Environment", *In Proceedings of Annual Workshop on Network and System Support for Games*, 2010, pp. 140-148.

[11] D. Horn, E. Cheslack-Postava, B.F.T. Mistree, T. Azim, J. Terrace, M.J. Freedman, P. Levis, "To Infinity and Not Beyond: Scaling Communication in Virtual Worlds with Meru", *Stanford Computer Science Technical Report*, CSTR 2010-01.

[12] Open Simulator, 2014, http://www.opensimulator.org.

[13] S. Yamamoto, Y. Murata, K. Yasumoto, M. Ito, "A Distributed Event Delivery Method with Load Balancing for MMORPG", *In Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, 2005, pp. 1-8.

[14] K. Lee, D. Lee, "A Scalable Dynamic Load Distribution Scheme for Multi-Server Distributed Virtual Environment Systems with Highly-Skewed User Distribution", *In Proceedings of ACM Symposium on Virtual Reality Software and Technology*, 2003, pp. 160-168.

[15] J. Jardin, D. Zappala, "A Hybrid Architecture for Massively Multiplayer Online Games", *In Proceedings of 7th ACM SIGCOMM Workshop on Network and System Support for Games*, 2004, pp. 60-65.

[16] B. De Vleeschauwer, B. Van Den Bossche, T. Verdickt, F. De Turck, B. Dhoedt, P. Demeester, "Dynamic Microcell Assignment for Massively Multiplayer Online Gaming", *In Proceedings of the 4th ACM SIGCOMM Workshop on Network and System Support for Games*, 2005, pp. 1-7.

[17] D. Ahmed, S. Shirmohammadi, "A Microcell Oriented Load Balancing Model for Collaborative Virtual Environments", *In Proceedings of IEEE Conference on VECIMS*, 2008, pp. 86-91.

[18] T. Debeauvais, A. Valadares, C.V. Lopes, "RCAT: A Scalable Architecture for Massively Multiuser Online Environments", 2013, http://www.ics.uci.edu/~tdebeauv/files/2013-RCAT.pdf.

[19] World of Warcraft, 2014, http://www.worldofwarcraft.com.

[20] M.T. Najaran, S.Y. Hu, N.C. Hutchinson, "SPEX: Scalable Spatial Publish/Subscribe for Distributed Virtual Worlds without Borders", *In Proceedings of 5th ACM Multimedia Systems Conference*, 2014, pp. 127-138.

[21] L. Ricci, E. Carlini, "Distributed Virtual Environments: From Client Server to Cloud and P2P Architectures", *In Proceedings of Int. Conf. on High Performance Computing and Simulation (HPCS)*, 2012, pp. 7-8.

[22] E. Carlini, M. Coppola, L. Ricci, "Integration of P2P and Clouds to Support Massively Multiuser Virtual Environments", *In Proceedings of the 9th Annual Workshop on Network and System Support for Games*, 2010, pp. 1-6.

[23] M.T. Najaran, C. Krasic, "Scaling Online Games with Adaptive Interest Management in The Cloud", *In Proceedings of NetGames'10*, 2010.

[24] L. Chan, J. Yong, J. Bai, B. Leong, R. Tan, "Hydra: A Massively-Multiplayer Peer-To-Peer Architecture for the Game Developer", *In Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*, 2007, pp. 37-42.

[25] R. Olanda, M. Perez, J.M. Orduna, "Hybrid P2P Schemes for Remote Terrain Interactive Visualization System", *Future Generation Computer Systems*, Vol. 29, No. 6, 2013, pp. 1522-1532.

[26] A. Chen, R.R. Muntz, "Peer Clustering: A Hybrid Approach to Distributed Virtual Environments", *In Proceedings of the 5th ACM SIGCOMM Workshop on Network and System Support for Games*, 2006, p. 11.

[27] K.C. Kim, I. Yeom, J. Lee, "HYMS: A Hybrid MMOG Server Architecture", *IEICE Transactions on Information and Systems*, Vol. E87, 2004, pp. 2706-2713.

[28] E. Buyukkaya, M. Abdallah, R. Cavagna, "Vorogame: A Hybrid P2P Architecture for Massively Multiplayer Games", *In Proceedings of CCNC*, 2009, pp. 1-5.

[29] M. Almashor, I. Khalil, Z. Tari, A.Y. Zomaya, "Automatic and Autonomous Load Management in Peer-to-Peer Virtual Environments", *IEEE Journal on Selected Areas in Communications*, Vol. 31, No. 9, 2013, pp. 310-324.

[30] E. Carlini, M. Coppola, L. Ricci, "Evaluating Compass Routing Based AOI-cast by MOGs Mobility Models", *In Proceedings of SIMUTools'11*, 2011, pp. 328-335.

[31] A. Denault, C. Ca~nas, J. Kienzle, B. Kemme, "Triangle-Based Obstacleaware Load Balancing for Massively Multiplayer Games", *In Proceedings of 10th Annual Workshop on Network and Systems Support for Games*, 2011, pp. 4:1-4:6.

[32] B. Van Den Bossche, B. De Vleeschauwer, T. Verdickt, F. De Turck, B. Dhoedt, P. Demeester, "Autonomic Microcell Assignment in Massively Distributed Online Virtual Environments", *Journal of Network and Computer Applications*, Vol. 32, No. 6, 2009, pp. 1242-1256.

[33] R. Ranjan, L. Zhao, "Peer-To-Peer Service Provisioning in Cloud Computing Environments", *The Journal of Supercomputing*, Vol. 65, No. 1, 2013, pp. 154-184.

[34] Sirikata, 2014, http://sirikata.com.