Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

COMPARISON OF THE EXPERIMENTAL AND SIMULATION RESULTS FOR DISTRIBUTED VIRTUAL ENVIRONMENTS APPLICATIONS FRAMEWORK

Xiaoyu Zhang
Denis Gračanin

Virginia Tech
2202 Kraft Drive
Blacksburg, VA 24060, USA

ABSTRACT

In our previous work we developed *Caffe Neve* framework that allows application developers to create flexible and extensible Distributed Virtual Environments (DVEs) applications from the distributed components. The components can serve as remote content sources that stream down interactive 3D content to the application integrator. Since DVEs are interactive, multi-user and networked systems, it is important to ensure good responsiveness and overall performances of the developed applications. We investigate the performance of the integrated applications to evaluate the framework capability. We conducted experiments and used OMNeT++ simulation tool to explore the scalability issues. We use various metrics such as the response latency and service load estimation to evaluate the framework performance.

1 INTRODUCTION

Online 3D applications can benefit from the flexible and extensible architecture built upon the Web platform. Integrating remote 3D resources as the content source can distribute the development effort among groups of developers. The development and maintenance cost is shared therefore constructing and upgrading the applications can be more efficient. We propose our framework *Caffe Neve* (Zhang and Gracanin 2008, Zhang and Gračanin 2008) as the standard approach for integrating distributed 3D resources to create flexible and extensible online 3D applications. The framework incorporates Web services (Alonso, Casati, Kuno, and Machiraju 2004, Hutchinson, Kotonya, Walkerdine, Sawyer, Dobson, and Onditi 2007) and ontology for application level integration, and uses streaming for improved performance.

Caffe Neve provides an effective support on broadcasting live 3D interactive content from the resource owners to the end users. Using *Caffe Neve* the distributed resource owners can develop their applications into distributed services. The services are the coarse-grained components to the framework. The application integrator creates the application content by consolidating the scene graph from various distributed components. The content updates are streamed from the service providers and the aggregated content is streamed to the clients. The framework defines the object relationships and application logic in the interaction model. When responding the client interactions, the service coordination is executed by following the interaction model rules.

In this paper we describe the framework architecture and demonstrate an example of integrating distributed resources to deliver live interactive 3D content. We evaluated the performance of the constructed applications. We simulate the framework behavior in a large scale in order to study the scalability of the framework, The evaluation results can be used as the reference for application developers when building performance critical applications.

2 RELATED WORK

The content of 3D applications is based on the 3D scenes and the virtual object behaviors corresponding to the interactions in the virtual world. Providing application compatible 3D models is the most common way of contributing application content. For example, third party developers can create the interactive 3D models for video games using the game level editors and the models can be seamlessly integrated into the games (Pipho 2002). Most online 3D applications use markup languages in 3D object modeling. The XML based descriptive language describes the geometry information as well as other metadata specified by applications for rendering. For example, KML (Keyhole Markup Language) is an XML-based file format for creating 3D geospatial visualization objects for Google Earth (Google 2011). Developers can develop and upload their KML files to share with other developers and users. Google Earth users can download online KML files and interact with the 3D models rendered by the application.

Virtual Environments (VEs) frameworks that use a component-oriented architecture provide a development and deployment environment to integrate an application from smaller units. They allow third-party developers to contribute implementations to the applications with much less development cost (Johnson 1997). VEs frameworks such as VRJuggler (Bierbaum, Just, Hartling, Meinert, Baker, and Cruz-Neira 2001), DIVERSE (Kelso, Satterfield, Arsenault, Ketchan, and Kriz 2003), and Avocado (Tramberend 1999) adapt component-oriented architecture, and third-party developers can use the frameworks to contribute the application implementations with much less development cost. Most frameworks support component integration at the system level, providing predefined API for interface coupling. For example, VRJuggler (Bierbaum, Just, Hartling, Meinert, Baker, and Cruz-Neira 2001) and DIVERSE (Kelso, Satterfield, Arsenault, Ketchan, and Kriz 2003) are created to build reusable components or provide the components that abstract the platform-dependent devices into platform-independent objects for different applications.

Web services technology is widely adapted by Internet applications for resource sharing and distributed component incorporation. Web services are self-contained and self-descriptive software units for web applications (Alonso, Casati, Kuno, and Machiraju 2004). Integrating 3D application content from services has been used in 3D visualization applications. Distributed services can implement more complex functions than a data repository. Pullen et al. proposed a framework called XMSF (Extensible Modeling and Simulation Framework) which expands HLA (High Level Architecture) using Web services technology to aggregate distributed simulations (Pullen, Brunton, Brutzman, Drake, Hieb, Morse, and Tolk 2005).

Constructing 3D applications from distributed components also requires an ensured performance for real-time applications. Delivering remote 3D data in real-time provides the support for 3D content broadcasting or podcasting. Ohta et al. (Ohta, Kitahara, Kameda, Ishikawa, and Koyama 2007) developed a live 3D video system that could deliver stream data to remote PCs. The live data was captured from different cameras. The 3D models and animations were dynamically constructed from the video. The remote clients played the 3D video by reconstructing the scene from the stream data. Creating 3D podcast content also has been researched by Nvidia (Fotouhi 2010). They creates a video broadcast environment for stereo 3D content. However, the technology depends on specific software and the 3D content can hardly get integrated in the shared virtual environment applications.

An enhancement in the architecture to improve the performance is to introduce peer-to-peer network of communication (Cavagna, Royan, Gioia, Bouville, Abdallah, and Buyukkaya 2009). Streaming large-scale 3D content between the nodes in the P2P networks can potentially accommodate more clients without jeopardizing the server performance (Hu 2006). Adapting streaming technique can resolve the request-response limit when introducing SOA in the framework. Streaming technology has been used in 3D applications for delivering 3D content and animation data (Hosseini and Georganas 2002). In addition, streaming technology has the capability of actively pushing the updates to the subscribers. Streaming Web server or Web services can support real-time pushing of new data to the clients using HTTP protocol (Comet) (Comet 2008).

3 CAFFE NEVE

Caffe Neve uses service oriented architecture (SOA) to integrate the third-party services (Zhang and Gracanin 2008, Zhang and Gračanin 2008). The seamless integration of a DVEs application using *Caffe Neve* is accomplished at both system level and semantic level. The system level integration ensures the data from the services is transferred and assembled promptly in the right format. The semantic level integration ensures the integrated application data represents the correct and complete application logic. Figure 1 shows the framework architecture and the three major components in the framework: *component services*, *container*, and *clients*. In this section, we highlight three architecture features of the framework.

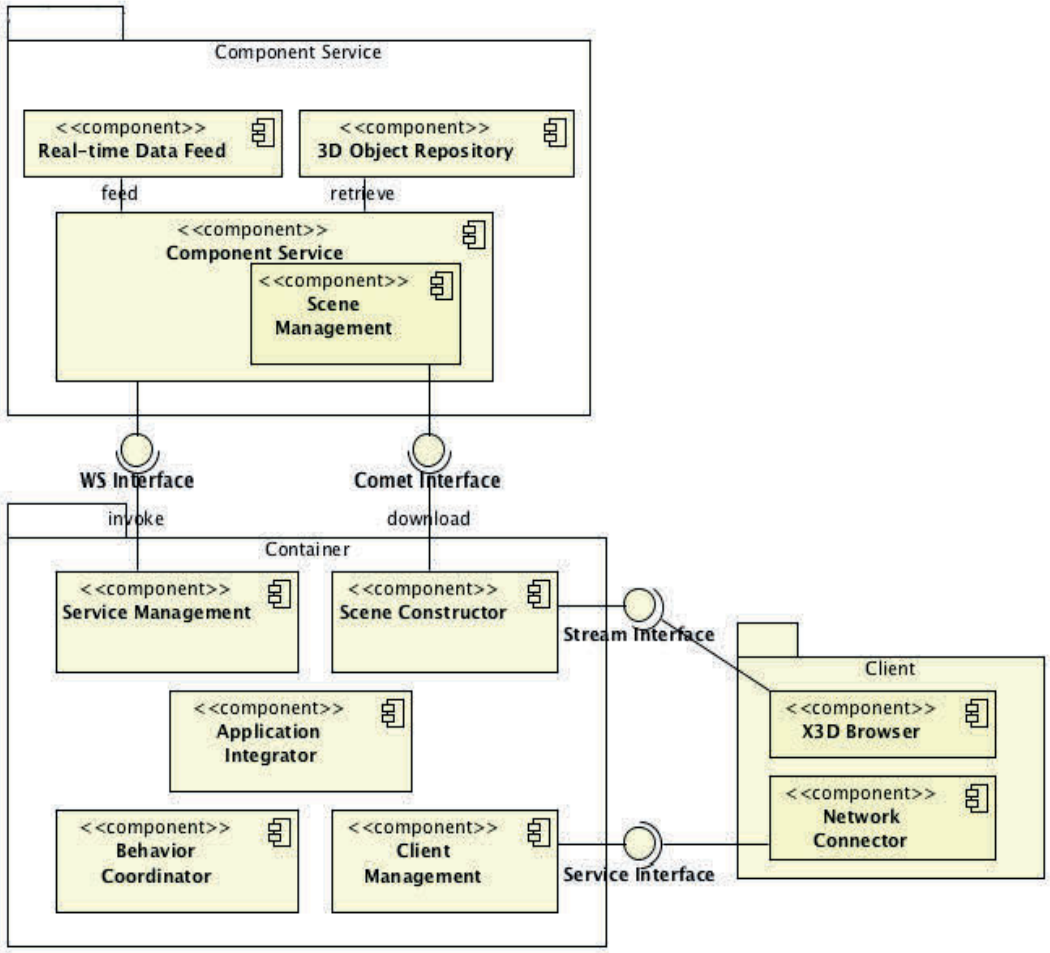

Figure 1: Framework Architecture.

3.1 Application Content in X3D

In general the application content of DVEs applications is presented in a scene graph. In the framework, eXtensible 3D (X3D) is used to describe the scene graph and the UI interactions. The application content in X3D can be delivered over the network. X3D is an ISO standard for XML-enabled 3D file format aimed to facilitate the interactively manipulating, communicating and displaying scenes (Web 3D Consortium

2011). The advantage of using X3D is that users can use a standard X3D browser to access any X3D applications.

Our framework dynamically generates interactive 3D user interface (UI) for the integrated application. Each service provider provides the description of the interactive object behaviors. The container integrates the application content from various providers and generates a consistent interactive 3D application. The scene data is created in X3D and delivered using MPEG-4 streaming format. The end users use standalone X3D browsers to access and interact with the application.

Users access the integrated application using 3D browsers or similar standalone clients. The framework delivers the constructed DVEs application to each client in a form of collaborative application content with interactive 3D UI. The dynamically generated 3D UI behaviors consistently on each part of the integrated application content regardless the sourcing contributors. The control of the multi-user interactions on the shared piece of application content is coordinated on the centralized application integrator of the framework.

3.2 Service Integration

We introduce ontology in the framework as a top-down approach for the component services integration. Framework level ontology is developed to abstract the interaction model in general VE applications. Based on the ontology, the semantic information for an application instance of a certain domain is modeled into what we call a *profile*. The profile is the agreement of the application content and interactions among the components providers. The terminology defined in the profile provide a vocabulary of the annotations that can be used to attach semantic information for the component services in the framework.

Component service developers use the profile terminology to encode the object behaviors as supplementary document for the service integration. In the profile, application events and the corresponding actions are also defined. The service coordination and application execution are triggered by collaborating and processing the runtime events. In this way, the framework combines the event-driven architecture in the SOA for the component coordinations.

3.3 Real-time Streaming

The real-time data is collected and animated from the service provider. Since the animation cannot be pre-defined in the application, the real-time animation data is streamed from the service provider to the application integrator which applies the animation in the application content. Each service provides two types of interfaces: Web service interface and Comet streaming interface (Comet 2008). The controls that manipulate the content are exposed via a Web service interface. The 3D application content from a service is delivered through the Comet streaming interface.

The component service developed by the third party developers are self-contained streaming Web services. The descriptive scene data is encoded using Web 3D standards (X3D/XMT/MPEG-4) and streamed to the container via a Comet streaming channel. The application integrator composes the component data into applications and streams down the application content to the end users.

4 SIMULATION

4.1 Real World Data Collection

We developed a reference implementation of *Caffe Neve* for proof of concept. To evaluate the framework performance, we created a 3D road traffic map application (Figure 2) by integrating five distributed services.

Among the five services, service A and B are two 3D map data providers. Service A provides the map data of the road, and service B provides the 3D building data of the same area in the map. Service C provides the road traffic data for the roads on the map. It delivers 3D shaped road traffic visualization data for the users who are approaching a Point of Interest (POI). Service D offers real-time traffic data using animations for the specific road intersections. The animation simulates the passing cars in the intersection

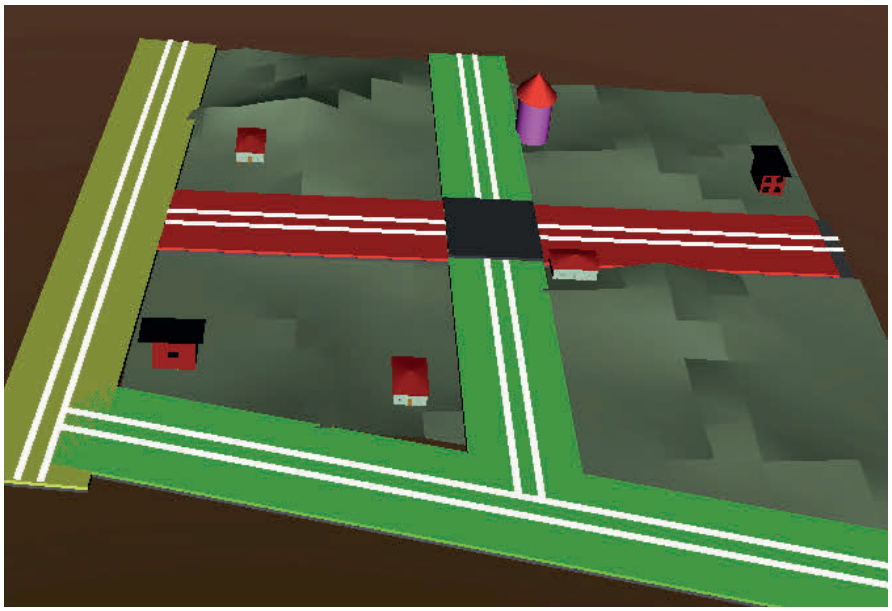

Figure 2: The 3D road traffic map application based on *Caffe Neve*.

at a certain time. Service E provides a navigation service. It will assist a user to navigate around when the user gets close to the POI. The five services are distributed on three remote servers located in Virginia — VA (Service A), California — CA (Services B and C), and Georgia — GA (Services D and E) across USA. The centralized container is located in Virginia.

We collect the performance data of the example application using the reference implementation. We conducted a series of experiments testing the network latency and response latency. The evaluation results are listed in Table 1. The results represent the typical communication features of the distributed components in our framework. The response latency is the time interval between the client side request and the response from the corresponding service. As our framework is not a one round trip client-server communication structure, the request-response interval involves more delay due to the message processing and relaying of the application integrator. The message processing time includes message parsing, XMT encoding/decoding, and MPEG-4 encoding/decoding. In our experiments, we recorded the response latency at the client when a user interacts on an interactive object. We made a comparison of the similar requests on the three servers and use the data as the reference for large scale simulation.

Table 1: The response latency on the servers.

Servers	Average Network Latency	Average Response Latency	Network Latency Percentage
GA Server	31ms	260ms	11.92%
VA Server	3ms	74ms	4.05%
CA Server	92ms	425ms	21.65%

4.2 Simulation Tool

In order to investigate the framework scalability and the component communication behavior, we need to extend the experiments to a larger scale. However, deploying services into various locations and tracking the behaviors of them can be very time-consuming. Collecting and processing data from a big number of

users also requires significant efforts. Therefore, we use a simulation tool to create a network environment for the framework behavior study. The simulation provides us the support to set up controlled experiments to leverage the influence of different factors to the framework performance.

The simulation tool we use is OMNeT++ (Varga and Hornig 2008). OMNeT++ is a discrete event network simulation framework. The framework is highly extensible due to its component-based modular architecture. Various simulation model (such as BitTorrent and HttpTools etc) and extended frameworks (such as INET and MiXiM etc) are available for users to set up or configure the well-tested network simulations.

4.3 Simulation Development

4.3.1 Modules

In the simulation, we created a network (called DVENetwork) with four simple modules: cloud, client, container, and component service. Cloud module is developed to simulate the Internet. The network topology is shown as in Figure 3. The network setup allows developers to easily configure the number of clients and services.

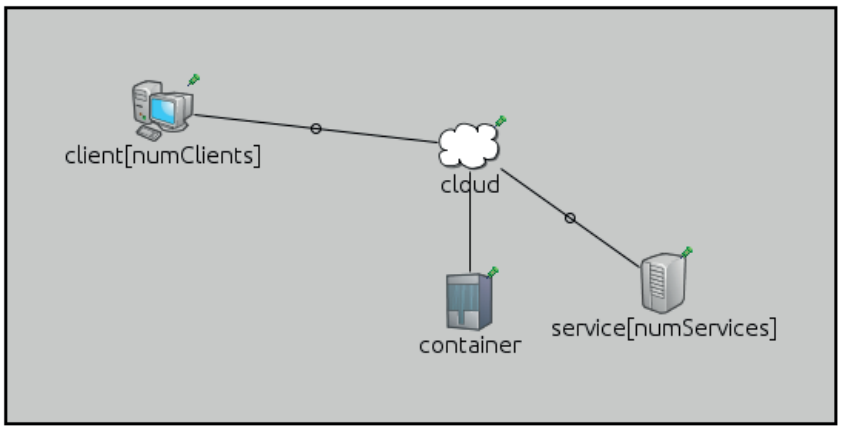

Figure 3: The configuration of the modules in the simulation network.

Figure 4a shows a running example of the network with three clients and two services. Figure 4b shows the same network setup with ten clients and six services configured. We easily use the same network setup to run our experiments with 600 users and 25 services or even more than that.

4.3.2 Messages

Messages in OMNeT++ represent events, network packets, and commands etc. The module starts execution whenever a message arrives. We created four message objects to simulate the network packets by extending the *cPacket* message class. Each message object is configured with different payload and scheduled delay duration.

SocketMsg is a message object which represents the network packets exchanging from the clients to the container. *WebServiceMsg* represents the network request and response between the container and the component services. *CometMsg* represents the stream data packets sent from the services to the container. *CometMsg* can be automatically triggered to send down to the container. *MPEG4Msg* represents the network messages sent from container to the clients. *MPEG4Msg* can be broadcasted to a group of clients to simulate the case when these clients are accessing the same virtual environment content. Figure 5 shows the message routing between the modules of three clients and two services.

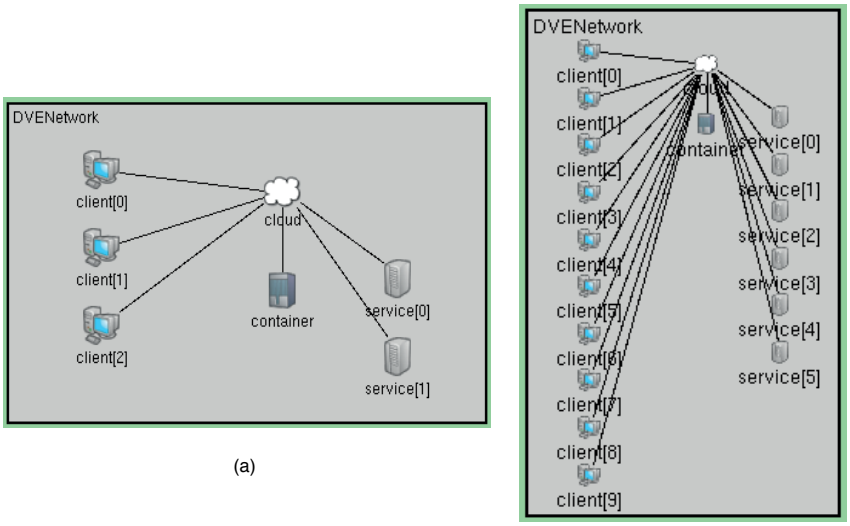

Figure 4: Runtime examples of the network.

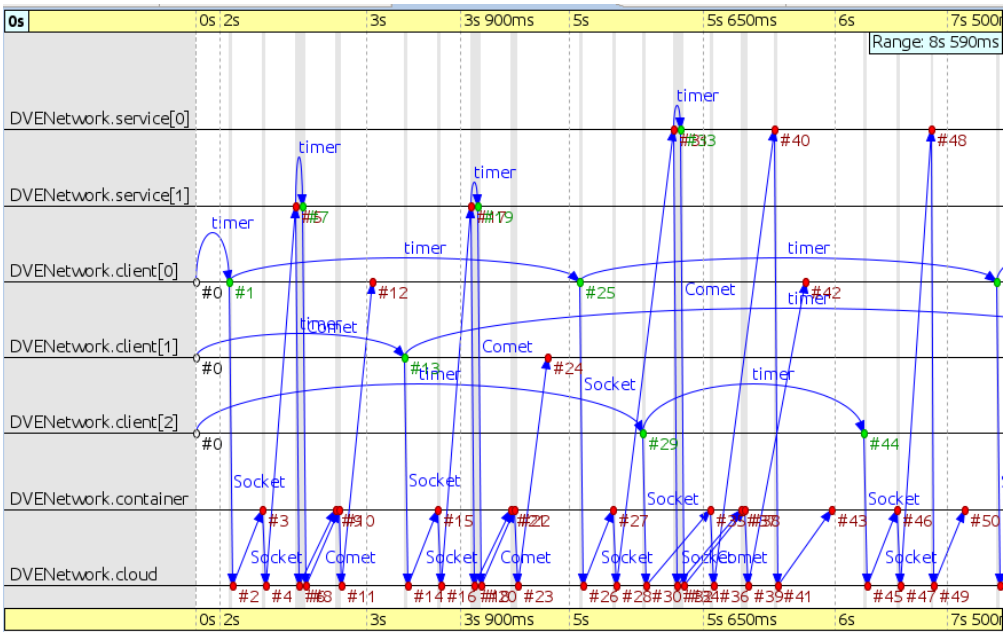

Figure 5: Message routing sequence chart.

4.4 Simulation Setup and Results

After running sets of experiments using the OMNeT++, we explored three interesting topics about the framework in an typical network environment. The first topic is still about the average response latency for clients. We set up the experiments by configuring the network delay (in the cloud module) and the process latency of the container and the services (in the container and component service modules). Both the network delay and the process latency use the exponential distribution with the mean values collected from the experiment results in Section 4.1. After running experiments with different number of clients and services, the response latency is fairly consistent for all of them. The average response latency is around 150ms to 200ms. The histogram in Figure 6 depicts the distribution of the response latency for a client when the framework has 300 clients and 20 services. The simulation duration of the experiment is 60s.

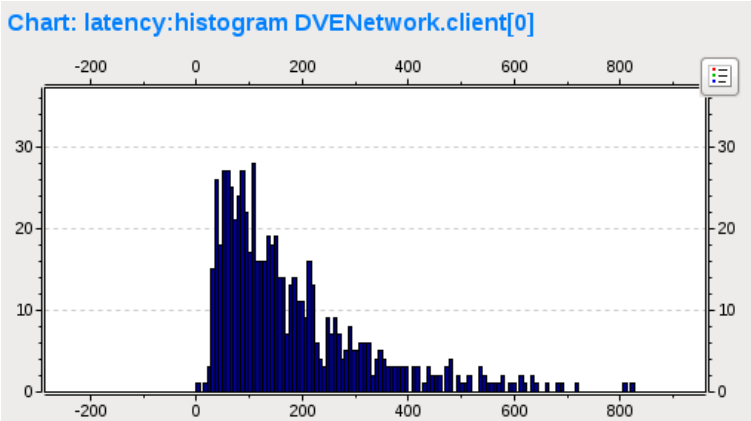

Figure 6: Histogram of response latency distribution.

The second topic is to study about the service load on the container when client number changes. The simulation results can be used to estimate the scalability of the framework when the server resources are limited. Figure 7 shows the number of messages arrived in 60s on the container comparing to a component service. There are 20 component services in the network and the client number increases from 100 to 600. The experiment assumes an ideal scenario that each service will selectively send the comet messages to the container per request.

The result illustrates that the container will have a huge number of messages to process when the client number increases. Not only the size of the messages from the clients get linearly increased but also the comet messages from the services. The services load gets increased as well, but not as significant as the container.

The third topic is to study the load of the service when the client number changes. The analysis of the simulation results can help us estimate the required processing capability of the service. In order to simulate the real world application, we created a simple model to simulate the service distribution and the user navigation behavior.

In our experiments, we configured 25 services. The services represent five by five geographically distributed spaces in the virtual world. Each service corresponds to the status change of the allocated space. The experiment starts with a number of users that are randomly placed in a virtual space. Within one minute all the requests from a client will be consistently forwarded to the same service. This simulates the user's interaction within the same virtual space. Once the time is up, the users will either stay in the same space or move to a neighbor space.

We carried out six experiments to evaluate the service load with different size of users. The service load on each server is estimated by counting the service invocation numbers received in 360s. The number of users ranged from 100 to 600. Table 2 summarizes the statistics collected from the six experiments and

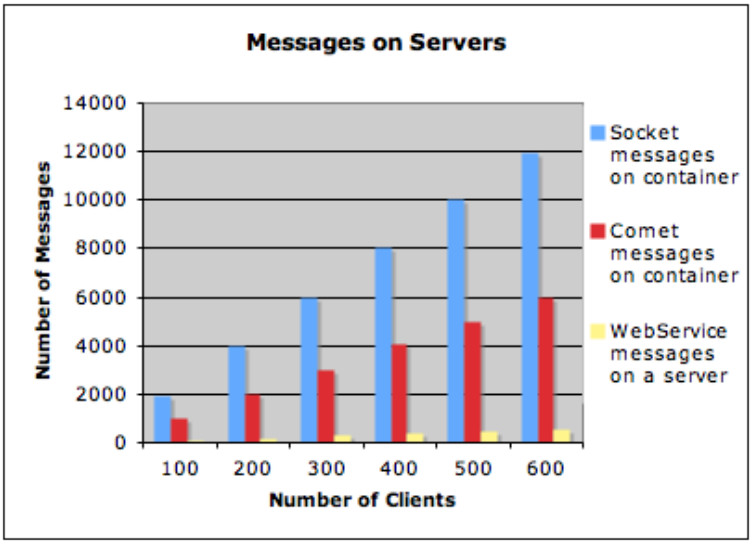

Figure 7: Messages arrived on servers.

Figure 8 shows the average service load variations with different number of users. The average service load is estimated by using the mean of the total service invocations on 25 services.

The linear increase of the service load indicates the ideal situation when each client sends similar amount of requests per minute. The data in Table 2 shows an interesting evidence that the service load is remarkably unbalanced among all the services. That means the user activities inside the virtual space are not equally distributed among the shared space. The service load on a popular service can be three times heaver than the average service load.

Table 2: The service load with different number of clients.

Number of users	MEAN	STDEV	MIN	MAX
100	405	356.8	71	1128
200	796	729.8	86	2342
300	1203	1092.7	132	3349
400	1604	1475.4	181	4322
500	2014	1775.4	312	5259
600	2409	2125.9	224	6281

In order to see how the service load distributed among the services, we generated histogram charts and compared the results of different user size. Figure 9 shows a comparison of the service load histogram chart of 100 users and that of 600 users. From the comparison, we can learn that the distribution pattern is similar regardless the number of users. Almost half of the services are below the average service load. This provides the application designers a useful information to distinguish the process capability requirement among the services when they have different popularity due to their virtual space locations.

5 DISCUSSION

In the simulation results, our data presents an ideal scenario when the container and the component services have unlimited computation capability. Meanwhile, the network latency is based on the data collected from the real world experiments. The real world network is more unpredictable than the simulated one due to the variations in network traffic. Therefore, we see the linear trends in the experiment results. However, that does not mean that the simulation results are too ideal to be useful. What we are interested in is to

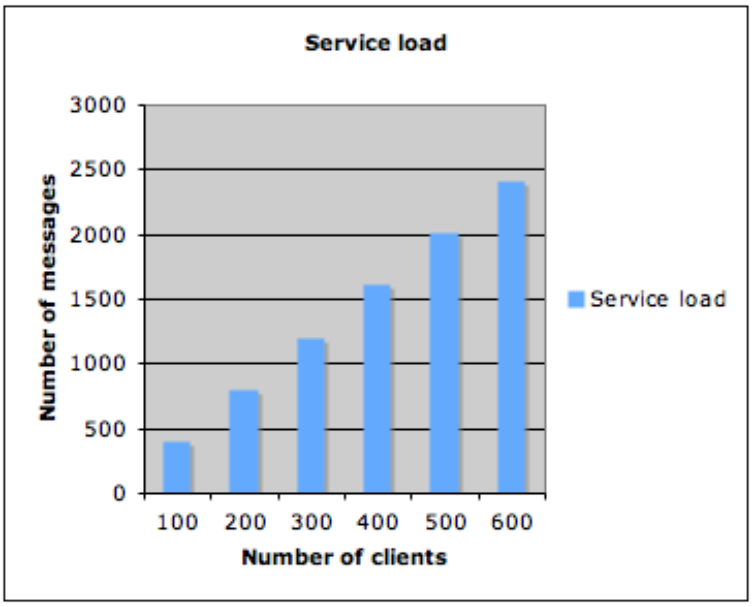

Figure 8: Service load variation with different number of clients.

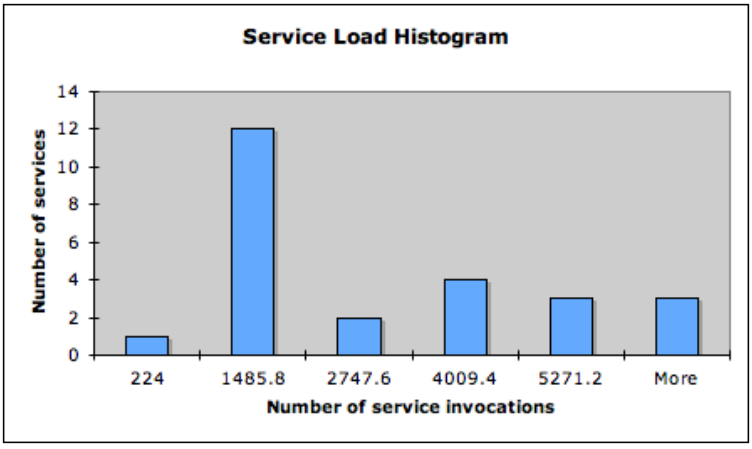

Figure 9: Comparison of service load distribution histogram.

explore the quantity of the network throughput and the service load impacts on the service and container. The simulation results reveals the scalability challenges the container and the possible processing capability requirements to the services for a specific number of users.

Another interesting topic worth exploring is how to more accurately simulate the user's behavior in the virtual world. In our current implementation the results already give useful information. In a more accurate model, we should collect user behavior data from typical DVEs applications (e.g. Second Life) and build the model into the simulation. In addition, in our simulation we divided the virtual space into five by five connected spaces and assigned each service to maintain the status change of the virtual content. A more accurate way to simulate the real world case is to divide the space in three dimensions and use approximate nearest neighbor searching algorithm to locate possible neighborhoods. To make it more similar to the actual DVEs applications, we can also introduce the priority to the services that have popular POIs.

6 CONCLUSION

In this article, we described *Caffe Neve*, an extensible and flexible framework for building distributed virtual environment applications. The framework uses service oriented architecture for application construction and streaming for improved performance. We developed a reference implementation to study the behaviors of the framework regarding to its network latency, service processing capability, and streaming performance. In order to study the framework performance in a larger scale, we use simulation tool OMNet++ to develop a network environment that simulates the running framework with different number of services and users. We modeled the users activities in the simulation and study the framework performance by changing the user numbers. In the future study, we will introduce a better user activity modeling based on real world data collection and add more network and server process capability constraints to improve the accuracy of the simulation results.

REFERENCES

Alonso, G., F. Casati, H. Kuno, and V. Machiraju. 2004. *Web Services: Concepts, Architecture and Applications*. Springer Verlag.

Bierbaum, A., C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. 2001. "VR Juggler: A Virtual Platform for Virtual Reality Application Development". In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)*, 89. Washington, DC, USA: IEEE Computer Society.

Cavagna, R., J. Royan, P. Gioia, C. Bouville, M. Abdallah, and E. Buyukkaya. 2009, January. "Peer-to-peer visualization of very large 3D landscape and city models using MPEG-4". *Image Commun.* 24:115–121.

Comet, Wikipedia 2008. "Comet (programming)". http://en.wikipedia.org/wiki/Comet_(programming) [Last accessed: Jan. 2011].

Fotouhi, F. 2010, july-sept.. "New Products". *Multimedia, IEEE* 17 (3): 79 –80.

Google 2011. "KML - Google Code". http://code.google.com/apis/kml/ [Last accessed: Feb. 2011].

Hosseini, M., and N. D. Georganas. 2002. "MPEG-4 BIFS streaming of large virtual environments and their animation on the web". In *Web3D '02: Proc. of the seventh international conference on 3D Web technology*, 19–25. New York: ACM.

Hu, S.-Y. 2006. "A case for 3D streaming on peer-to-peer networks". In *Proceedings of the eleventh international conference on 3D web technology*, edited by D. Gračanin, Web3D '06, 57–63. New York, NY, USA: ACM.

Hutchinson, J., G. Kotonya, J. Walkerdine, P. Sawyer, G. Dobson, and V. Onditi. 2007. "Evolving Existing Systems to Service-Oriented Architectures: Perspective and Challenges". In *Proceedings of the 2007 IEEE International Conference on Web Services (ICWS 2007)*, edited by L. jie Zhang, K. P. Birman, and J. Zhang, 896–903.

Johnson, R. E. 1997. "Frameworks = (components + patterns)". *Commun. ACM* 40 (10): 39–42.

Kelso, J., S. G. Satterfield, L. E. Arsenault, P. M. Ketchan, and R. D. Kriz. 2003. "DIVERSE: a framework for building extensible and reconfigurable device-independent virtual environments and distributed asynchronous simulations". *Presence: Teleoper. Virtual Environ.* 12 (1): 19–36.

Ohta, Y., I. Kitahara, Y. Kameda, H. Ishikawa, and T. Koyama. 2007, October. "Live 3D Video in Soccer Stadium". *Int. J. Comput. Vision* 75:173–187.

Pipho, E. 2002. *Focus on 3D Models*. Premier Press.

Pullen, J. M., R. Brunton, D. Brutzman, D. Drake, M. Hieb, K. L. Morse, and A. Tolk. 2005. "Using Web Services to Integrate Heterogeneous Simulations in a Grid Environment". *Future Generation Computer Systems* 21:97–106.

Tramberend, H. 1999. "Avocado: A Distributed Virtual Reality Framework". In *VR '99: Proceedings of the IEEE Virtual Reality*, edited by L. Rosenblum, P. Astheimer, and D. Teichmann, 14. Washington, DC, USA: IEEE Computer Society.

Varga, A., and R. Hornig. 2008. "An overview of the OMNeT++ simulation environment". In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, edited by S. Molnár, J. R. Heath, O. Dalle, and G. A. Wainer, Simutools '08, 60:1–60:10. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Web 3D Consortium 2011. "Communicating with real-time 3D across applications, networks, and XML Web services". http://www.web3d.org/ [Last accessed: July 2011].

Zhang, X., and D. Gracanin. 2008, December. "Service-oriented-architecture based framework for multi-user virtual environments". In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Moench, O. Rose, T. Jefferson, and J. W. Fowler, WSC '08, 1139–1147. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Zhang, X., and D. Gračanin. 2008. "Streaming web services for 3D portal applications". In *Web3D '08: Proceedings of the 13th international symposium on 3D web technology*, edited by D. W. Fellner and S. Collins, 23–26. New York, NY, USA: ACM.

AUTHOR BIOGRAPHIES

XIAOYU ZHANG is a PhD student in the Department of Computer Science at Virginia Tech. His research focuses on the distributed virtual environments, Web 3D, and computer supported cooperative work. His email address is zhangxy@vt.edu.

DENIS GRAČANIN is an Associate Professor in the Department of Computer Science at Virginia Tech. His research interests include virtual reality and distributed simulation. He is a senior member of ACM and IEEE and a member of AAAI, APS, ASEE and SIAM. His email address is gracanin@vt.edu.