RESTful Massively Multi-User Virtual Environments: A Feasibility Study

Cristina V. Lopes, Thomas Debeauvais, Arthur Valadares
University of California, Irvine
{lopes, tdebeauv, avaladar} @ics.uci.edu

***Abstract*—The increased bandwidth made available to consumers in the past decade has enabled the spread of Massively Multi-user Virtual Environments (MMVEs). MMVEs require fast and frequent updates to be delivered to a considerable number of users. To address these difficult requirements, developers resort to network architectures tightly coupling clients and servers. While providing a good enough quality of experience, these architectures often compromise scalability and fault tolerance, and limit the design of user interactions and gameplay.**

To gain scalability and fault tolerance back, we investigate the possibility of developing MMVEs using the REST principles. Our theoretical and experimental results show that MMVEs can be built in a REST style, providing that new data management systems are developed.

***Index Terms*—MMVE, MMO, scalability, client-server, architecture.**

I. Introduction

Massively Multi-User Virtual Environments (MMVEs) such as World of Warcraft or Second Life are characterized by their support of multi-user interactions at a large-scale: thousands of users engage in several quasi-real-time pair-wise and group-wise interactions. The scale of MMVEs greatly impacts the responsiveness and consistency of the game world perceived by the players. The more users and the more user-generated events to be shared, the more difficult it is to support smooth interactions, and this difficulty increases not linearly but quadratically with the number of users [1].

Because of the demands for responsiveness, MMVEs are engineered using all sorts of optimizations. While resulting in efficient servers, these optimizations hamper scalability and fault tolerance, and result in one-of-a-kind systems that are hard to maintain and evolve. There have been very few general principles in MMVE architectures, as each application ends up being specifically-engineered for its unique requirements.

The last 15 years have seen the increasing success of MMVEs, but also of the Web. Unlike MMVEs, web applications follow a foundational set of principles, or architectural style, known as REST [2]. One of the REST principles is statelessness of the application servers: any state that needs to be preserved between two client requests is supposed to be persisted to a database. When this principle is followed, scalability and fault tolerance are achieved simply by adding hardware executing the same application program. This is known as horizontal scalability, or "scale out." Architectures in a REST style are particularly suitable for cloud computing infrastructures, as servers can be added on the fly.

In this paper, we explore the idea of architecting MMVEs in a REST style. This idea embeds several conflicting forces. First, although the online game engineering community has long been using client-server architectures, it has kept away from the Web, with some claiming that "many problems faced by multiplayer games cannot be solved using REST techniques." [3]. Second, the use of HTML5, including Websockets and WebGL, is showing serious opportunities for MMVE with web-based clients. While MMVEs in web browsers will certainly happen, the engineering of the server-side and the suitability of REST principles remain open questions.

Our analysis has a concrete scaling objective in sight. It is believed that the maximum number of directly interacting users ever achieved for a MMVE is 1,600 in the game EVE Online[1]. In World of Warcraft that number is around 120[2], and in Second Life it is around 80. Our study focuses on whether it is possible to achieve the level of multi-user interactivity of EVE Online following the REST principles.

This paper does not contribute any particular solution; instead, its contribution is the formulation and analysis of a design problem. REST principles are highly desirable for scalability but they conflict with performance. The goal of this work is to thoroughly understand the weaknesses of REST for MMVEs, so that we can then begin to design solutions to mitigate those weaknesses.

In [4], we described a potential architectural design for a RESTful MMVE that we call RMVE. In this paper, through a theoretical bandwidth analysis, we identify the hardware and network required to support 1,600 interacting users. We follow with an experimental analysis illustrating some bottlenecks and limitations of RMVE, and unveil new opportunities for tackling those problems.

II. Bandwidth Theoretical Analysis

Before implementing any prototype, we can first estimate whether this architecture realistically can support 1,600 players interacting simultaneously in quasi-real-time. If so, what is the order of magnitude of proxies and application servers needed to support them? We assume a worst-case scenario where all the players continuously move, and all the players want to know the position of all the other players.

[1]As reported in a forum post by one of the engineers http://www.eveonline.com/ingameboard.asp?a=topic\&threadID=1446750

[2]As seen on the WoW wiki http://www.wowwiki.com/Wintergrasp#Queuing

Variable	Symbol	Value
Backend bandwidth	B	1 Gbps
Client identifier length	c	4 bytes
Client downstream	C_{down}	15 Mbps
Client upstream	C_{up}	2 Mbps
Frequency of updates	f	20 per sec
Packet header size	H	80 bytes
Message length	L	30 bytes
Backend messages per frame	M	47
Ethernet MTU	MTU	1500 bytes
Number of clients	N	1,600
Number of proxies	P	TBD
Number of app servers	S	TBD

TABLE I
VARIABLES USED IN THE THEORETICAL ANALYSIS, SORTED BY ALPHABETICAL ORDER OF THEIR SYMBOL.

We iterate over the eight data streams within the system: 1) Client's upstream to proxies. 2) Proxy's downstream from clients. 3) Proxy's upstream to game servers. And so on, down to 8) Client's downstream from proxies. Analysis parameters and values are shown in Table I. We assume a DSL line with a downstream of $C_{down} = 15$ Mbps and upstream of $C_{up} = 2$ Mbps.[3] The frequency of updates must be sufficiently high to allow for To achieve real-time user interactions, the clients send message updates at a frequency of $f = 20$ per second.

We also assume that the messages sent by the client contain around $H = 80$ bytes of headers and $L = 30$ bytes of payload. To simplify our analysis, once the proxy receives a client message, it forwards it without adding metadata such as timestamp or origin to an application server in a round-robin fashion. Therefore, the messages sent by the proxy to a server also contain $L = 30$ bytes of data. Similarly, we assume the game server reply to contain $L = 30$ bytes of data to simplify our analysis.

Given the Ethernet MTU, the proxies and servers can pack at most $M = \lfloor \frac{MTU-H}{L} \rfloor = 47$ messages per Ethernet frames. In the remaining of the analysis, we list the constraints that our system must satisfy.

Step 1) Each client sends:

$$(L+H).f = 17kbps \ll C_{up} = 2Mbps \tag{1}$$

Step 2) Each proxy receives from all its clients:

$$\frac{N}{P}.(L+H).f < P_{down} = 1Gbps \tag{2}$$

For $N = 1,600$, this gives $P < 1$: a single proxy is enough.

Step 3) A proxy receives $\frac{N.f}{P}$ messages per second from all the clients, and has to forward $\frac{N.f}{P.M}$ frames per second to the server layer. Those frames contain $H + M.L$ bytes. Hence each proxy sends to the server layer:

$$\frac{N}{P}\frac{f}{M}(H + M.L) < P_{up} = 1Gbps \tag{3}$$

This gives $P < 1$ again.

[3]As seen in the high-end Comcast Internet offers, http://www.comcast.com/Corporate/Learn/highspeedinternet/highspeedinternet.html

Step 4) To distribute the load, proxies send messages to game servers in a round-robin fashion. Each server receives:

$$\frac{N}{S}\frac{f}{M}(H + M.L) < S_{down} = 1Gbps \tag{4}$$

For $N = 1,600$, this gives $S < 1$.

Step 5) Each server receives $\frac{N.f}{S}$ messages per second. For each of those messages, it forwards the same message (L bytes) and a list of clients targeted by this message ($c.N$ bytes). This whole reply requires $\lceil \frac{L+c.N}{MTU-H} \rceil = 5$ frames for 1,600 clients. In the end, each server sends to the proxy layer:

$$\frac{N.f}{S}(L + c.N + 5.H) < S_{up} = 1Gbps \tag{5}$$

Accommodating 1,600 clients requires 2 game servers.

Step 6) Each proxy receives what all the servers are sending:

$$N.f(L + c.N + 5.H) < P_{down} = 1Gbps \tag{6}$$

Note that this equation only depends on the number of clients, N, not on the number of application servers, S, or the number of proxies, P. As it is, this equation is only valid up to N = 1,197. To accomodate N = 1,600 clients, we need to provide each server its own LAN to which each proxy connects to.

Step 7) Each proxy sends to its connected clients:

$$\frac{N}{P}N.f\frac{H + M.L}{M} < P_{up} = 1Gbps \tag{7}$$

A scenario with 1,600 clients requires 13 proxies.

Step 8) Each client downloads from its proxy:

$$N.f.\frac{(H + M.L)}{M} = 8.1Mbps < C_{down} = 15Mbps \tag{8}$$

Conclusion: From a bandwidth perspective, each of the 1,600 clients needs a modern DSL connection, and the back-end requires 13 proxies and 2 game servers. However, our calculations ignored that complex game logic will require more CPU power, and therefore likely more than a couple servers.

III. Experimental Analysis

A. Setup

We implemented a proxy and a game server in C#[4], and used MySQL for the database. Our clients, implemented in Java and Javascript, use websockets to connect to the proxy. The proxy and game server communicate via TCP. We used commodity machines: quad-core Intel i7 HT with 8GB RAM for the proxy, server, and database. All the machines ran on Windows 7 and shared a 1Gbps connection through a switch. The round-trip time from proxy reception of a client message to proxy forwarding of a server message was measured every second. We report its average over each 3-minute run.

We simulated players using 5 to 50 bots sending position messages 20 times per second. When receiving a client message, the proxy forwards it to the game server, which stores the player's new position in the database, retrieves the list of

[4]Our source code is available at https://github.com/gentimouton/rcat-gs

clients to notify (in our case: everyone), and sends back to the proxy the list of clients and the message to relay. We ran five different configurations. In the **PS** configuration, the proxy and server are on the same machine, communicate through loopback, and there is no database (the server updates positions in RAM). In **PSD**, we add a database on the same machine. When compared to PS, PSD shows the impact of the database without any network overhead. **PScD** resembles PSD, except the game server caches the list of users. **P-SD** is based on PSD too, except the proxy has been moved to another machine. Finally, in **P-S-D** the proxy, server, and database are on three different machines; this is the typical RMVE architecture. The last two scenarios highlight the impact of the network on performance.

B. *How does the theoretical analysis hold in practice?*

Fig. 1 confirms the quadratic increase in bandwidth from the proxy to all the clients, found in equation 7. This underlines the need for a client manager layer as a way to separate that quadratic network load from the rest of the system.

C. *RMVE overhead*

Fig. 2 shows the poor performance of RMVE (configuration P-S-D) compared to PS. For 50 clients, the backend round-trip time of P-S-D ($\mu = 59ms$, $99^{th}percentile = 330ms$) is an order of magnitude larger than the round-trip time of PS ($\mu = 3.2ms$, $99^{th}percentile = 19ms$).

Two factors explain this poorer performance. First, comparing P-SD to PSD shows that the network overhead between server and database can cause up to 30ms extra latency for 50 clients. Second, comparing PSD to PS shows that even without any network overhead, the database adds 10 to 15ms of latency for 50 clients. This shows the need for a distributed data management component faster than a relational database.

D. *Cache improvements*

As seen in Fig. 2, P-Sc-D is bound between P-S-D and PS. Not having to request the list of clients for each message received saves the application a round-trip to the database. Compared to P-S-D, P-Sc-D reduces the latency by 13.5 ms on average.

Moreover, Fig. 3 presents the bandwidth requirements of the data layer. In the PSD configuration, the communication from the server to the database grows linearly with the number of clients, but communication from database to server grows quadratically. That quadratical growth is due to the constant requests for the list of all clients. As the number of clients increases, an increasingly longer list is returned increasingly more frequently. Caching this list of clients, as shown in configuration PScD, tames the quadratic bandwidth increase into a linear one.

This pinpoints the critical role of caches in the realization of RMVE. MMVE-specific caches running at the application-level increase the application's complexity, may hamper scalability, and can introduce bugs. Therefore, we see the need for a caching component specifically designed for MMVEs, but in a general-purpose manner such as COSAR [5].

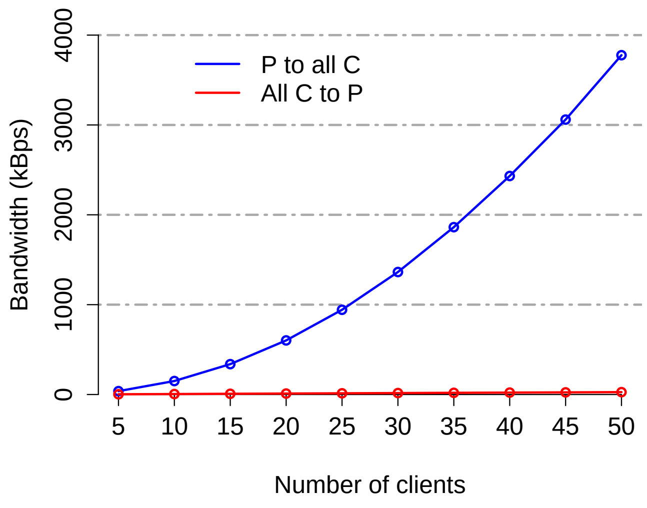

Fig. 1. Client to proxy bandwidth (as measured from scenario PS).

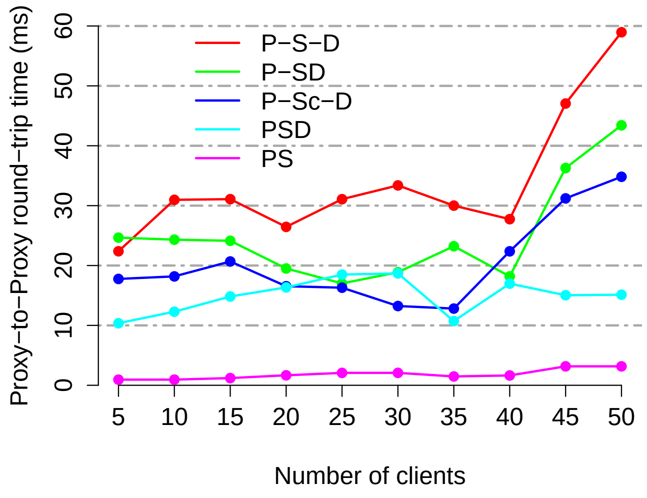

Fig. 2. Round-trip time for various configurations.

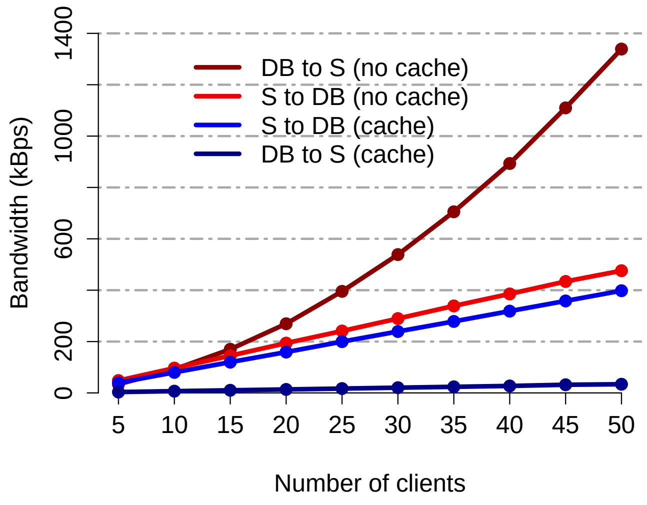

Fig. 3. Bandwidth between database and server without cache (PSD) and with cache (PScD).

IV. Related Work

MMVEs are only now within the realm of possibility for the Web platform. Social networks like Twitter and Facebook show some traits of MMVEs, with their very fast distribution of their users' posts among user slices of interest. To do so, web developers have been using Flash sockets, AJAX/Comet long-polling, and more recently Websockets.

MMVEs so far have been developed in parallel to the Web, having flourished primarily for online games and large simulation environments. The field is divided into two architectural camps: peer-to-peer (P2P) and client-server.

In P2P systems, the program that the user drives is both the simulator and the user interface. An additional network layer allows several of these peers to join in one logical simulation, where physical simulation of different parts of the scene happens in the different peers. Examples of P2P MMVEs and MMVE technologies include MiMaze [6], the High-Level Architecture standard [7], Unity3D [8], and many others.

Commercial MMVEs follow a client-server architecture. Their internal architectures vary considerably, but they all share the existence of one authoritative server-side to which clients connect. Besides the well-known commercial MMVEs such as Second Life and World of Warcraft, server-side systems aiming for scalability include the online game EVE Online and the open-source MMVE frameworks RedDwarf and Sirikata [9].

In EVE Online [10], clients connect to proxies, and servers are the bridge between the database and the proxies. Each server is in charge of one or multiple regions of the world. This implies that zones with thousands of clients become overloaded, and either reduce the level of interactivity and the frequency of updates [11], or crash. The most active "solar systems" of EVE Online can reach 1,600 clients before lag becomes apparent and gameplay severely altered[5].

RedDwarf [12] is an application server for MMVEs that provides a central data layer synchronized across servers. This data layer is accessed by the game logic layer through tasks. Similar to database transactions, RedDwarf tasks have ACID (Atomic, Consistent, Isolated, Durable) properties.

V. Discussion and Conclusions

MMVEs have challenging requirements, as they have to process fast-paced events from many clients, and persist the results of these events consistently. Because of its well-known performance weaknesses, REST has been largely dismissed from MMVE engineering. But REST principles, and statelessness in particular, are highly desirable for scalability.

As shown in the paper through RMVE, REST-based MMVEs have bandwidth bottlenecks and latency issues. But as with any REST application, RMVE also allows for caching data at several levels. What to cache, and for how long, depends entirely on the application logic. The list of connected players, for instance, changes relatively unfrequently and can be cached.

[5]EVE Online Dev blog "Introducing Time Dilation," http://www.eveonline.com/devblog.asp?a=blog&bid=900

The theoretical analysis tells us that the 4-layer REST architecture can reasonably match the highest reported level so far of multi-user interactivity – 1,600 mutually-interacting users – provided that some interest management schemes are in place. This is encouraging to experimentally test the scaling of RMVE.

The REST architecture, by itself, is not the source of high latency; the main problem is that ordinary implementations of REST for Web applications do not meet the demands of MMVEs. This is also encouraging, and points to the need of novel approaches for the design of storage/cache systems that handle streams of rapidly changing data better. Application developers should not have to worry about caching, as that is the recipe for entanglements that make applications unscalable and unruly. Instead, caches should belong to the data management layer and, as such, should present a uniform and powerful interface for application programmers to express the synchronization needs of their data.

Acknowledgments

This work was partially supported by NSF grant IIS-0808783.

References

[1] H. Liu, M. Bowman, R. Adams, J. Hurliman, and D. Lake, "Scaling virtual worlds: Simulation requirements and challenges." in *Winter Simulation Conference'10*, 2010.

[2] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[3] J. Watte, "REST and games don't mix," http://engineering.imvu.com/2010/12/18/rest-and-games-dont-mix/.

[4] T. Debeauvais, A. Valadares, and C. V. Lopes, "RCAT: a RESTful Client-server Architecture using TCP," in *Proceedings of the 10th Annual Workshop on Network and Systems Support for Games*, ser. Netgames '11, 2011.

[5] S. Ghandeharizadeh, J. Yap, S. Kalra, J. Gonzalez, E. Keshavarzian, I. Shvager, F. Carino, and E. Alwagait, "Cosar: A precis cache manager," USC, Tech. Rep., 2009, http://dblab.usc.edu/users/papers/cosar09.pdf.

[6] L. Gautier and C. Diot, "Design and evaluation of MiMaze, a multi-player game on the internet," 1998.

[7] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall, 1999.

[8] Unity3D, "Manual for networked multiplayer," http://unity3d.com/support/documentation/Manual/Networked+Multiplayer.html, 2010.

[9] D. Horn, E. Cheslack-Postava, T. Azim, M. J. Freedman, and P. Levis, "Scaling virtual worlds with a physical metaphor," *IEEE Pervasive Computing*, vol. 8, July 2009.

[10] D. H. Brandt, "Scaling EVE Online, under the hood of the network layer," CCP Games, Tech. Rep., 2005.

[11] Veritas, "Introducing time dilatation," http://www.eveonline.com/devblog.asp?a=blog\&bid=900, 2011.

[12] "Red Dwarf," http://www.reddwarfserver.org.