RCAT: a RESTful Client-scalable ArchiTecture

Thomas Debeauvais, Arthur Valadares, Cristina V. Lopes
School of Information and Computer Science,
University of California, Irvine
Irvine, CA, USA
{tdebeauv, avaladar, lopes}@ics.uci.edu

***Abstract*—In interactive multi-user virtual environments, the computation and bandwidth required to perform real-time simulation increases quadratically with the number of users. Handling communication, storage and computation resources on the same machine hinders scalability. This paper presents a system architecture abiding to the constraints of the REpresentational State Transfer (REST) architectural style. This architecture divides the resource requirements into separate layers and suggests how they can be addressed efficiently in isolation. Theoretical results show how the architecture scales with the number of clients.**

***Index Terms*—software architecture, virtual world, online games, scalability, REST**

I. Introduction

Most current massively multi-user environment (MMVE) can host up to a few hundred users interacting with each other. State-of-the-art approaches such as interest management or sharding work well in scenarios where many users do not need to see each other. These techniques are not enough for scenarios involving thousands of concurrent users potentially all interacting with each other in real-time: virtual stadium matches, concerts, or conferences.

This paper proposes an MMVE architectural style addressing the problems raised by the simulator-centric monolithic architecture [1]. Our architecture is based on the fact that the game logic is independent of the game data, an approach mentioned by previous works [1] [2]. We propose to go one step further and borrow an architectural style that made the Web scalable: REpresentational State Tranfer (REST) [3] and its five architectural constraints:

- **Client-server**: The state of the system is stored on the server-side, not on the client-side. This constraint excludes peer-to-peer MMVE solutions. Clients issue requests to a server resource, identified by a URL.
- **Stateless**: No client-specific information (*i.e.* context) can be stored on the server between two client requests. In the case of MMVE, our intuition is that an architecture in which the server-side processing machines contain state can not scale because of the overhead of data synchronization between those machines. The state has to be externalized from the data processing locus. The ACID properties of the tasks of Darkstar/RedDwarf make it a successful stateless data-synchronization layer for MMVE [4].
- **Cache**: Caches are intermediaries between system components. Their use is highly recommended, since they can reduce computations and traffic considerably. Caches contain information that is not supposed to change frequently. In the case of MMVE, the user position is not cacheable, but the list of current users could be.
- **Uniform interface**: A RESTful server interface should be able to receive messages with different formats. Metadata should accompany the actual data to help the system understand the message content. MIME-types or timestamps are examples of metadata that apply to MMVEs.
- **Layered system**: hardware proxies, load balancers, and software-level intermediaries are recommended. The first layered architectures for MMVEs appeared more than a decade ago.

II. RCAT Architecture

Figure 1 describes our architecture. The first tier consists of proxies, also called client managers or communication-intensive components in [1]. Proxies are responsible for routing messages from clients to servers and vice-versa. Clients send their messages to their proxy, and only receive updates when a client has performed an action. The proxies require a very high level of I/O operations, which results in a high CPU and network interface usage. Scripts send and receive messages like human clients. Time-based events are triggered by the script engine clock, not by the servers.

The "servant" tier is responsible for running the game logic and enforcing the game rules. Servants receive client information through the proxies of the first layer, and update the database accordingly. Servants are CPU-intensive components that may be suitable for high-end machines optimized for calculations. A new servant can be added on the fly at no synchronization cost. It only needs to notify the proxies of its presence. Proxies send messages to the servants they know in a round-robin fashion, distributing the load evenly between servants. The physics engine is a particular servant that reads from and writes into the database periodically, but does not receive messages from the proxies. The physics engine's heartbeat (if any) is independent of the rest of the system.

The data layer focuses on data manipulation and isolates the data synchronization problem into a single conceptual entity, be it a single or distributed database. What distinguishes our architecture to current MMVE approaches is an entire isolation of the state in a layer of its own. As such, domain-independent database techniques and optimizations in parallel

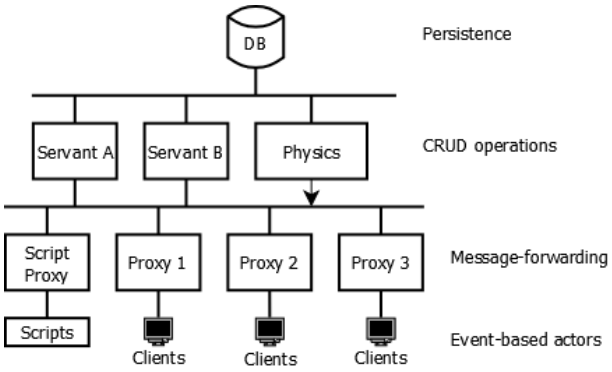

Fig. 1. RCAT architecture

data processing and synchronization can be used.

III. Theoretical Limits

We focus on a scenario where all users need to know each other's movements. Clients send a position message of 30 bytes of data (and 80 bytes of headers) to the system 20 times per second. We simplify the analysis by assuming a single proxy and a single servant on separate machines.

A. Client-Proxy

If n clients share a bi-directional 100-Mbps connection to the proxy, each of them sends $20 \times 110 = 2200$ Bytes per second (Bps) and receives $n \times 20 \times 110 = 2200 \times n$ Bps. The maximum number of clients that can share this bi-directional connection satisfies $(2200 \times n) \times n \times 8 \leq 100.10^6$. That is, $n \leq 75$. In other words, for each proxy added to the system, at most 75 clients can be accommodated. In practice, however, collisions between packets happen before that threshold.

B. Proxy-Servant

For every client message it receives, the proxy has to send a message containing the same information (30 Bytes) as well as the client id (4 Bytes) to the servant. Adding 80 Bytes of header, the proxy sends $(34 + 80) \times n$ Bps to the servant. For each message it receives, the servant replies with the list of all connected clients' id ($4 \times n$ Bytes) and the position to broadcast (30 Bytes). Therefore, the servant sends $(4 \times n + 30 + 80) \times n$ Bps to the proxy. Roughly 1750 clients can be accommodated with a 100Mbps bi-directional proxy-servant connection.

C. Servant-Database

Let us first examine the total bandwidth required from the servant layer to the database. The servant asks the database to update the position field of a particular user only if that user's last-seen action timestamp field in the database is lower than the timestamp of the current message. Therefore, the query length is relatively long: around 200 Bytes. The servant also has to retrieve the list of all clients, because servants are stateless. This extra query is 50-Byte long. Each time a servant receives a message from a proxy, *e.g.* 20 times per second and per client, it has to perform those two requests. With n clients, those requests require $20 \times n \times (200 + 50) = 40n$ kBps. A 100Mbps link will be saturated after 2500 clients.

The list of clients is retrieved 20 times per second and per client. Each client information fits in 8 Bytes (one integer for IP and one for port), therefore the query answer measures $n \times 8$ Bytes. With n clients, the network has to support $20 \times n \times (n \times 8) = 160n^2$ kBps, that is 280 clients for a 100-Mbps link.

IV. Discussion and Conclusion

In our early experiments, we focus on the worst case scenario to observe where bottlenecks can arise in the architecture. In this scenario, all of the 50 clients have to send their position 20 times per second, and receive messages from all other clients. Results so far indicate that the proxy's CPU consumption increases linearly with the number of clients, and reaches 15% on a quad-core Intel i7 HT. The client-proxy connection may, therefore, be the limiting factor for the proxy.

A pure event-based approach may not be the most efficient to handle communication in our system. Currently, each TCP message exchanged between proxy and clients contains 30 bytes of actual data for 80 bytes of header, resulting in a protocol efficiency of $30/(30+80) = 27\%$. Bucket synchronization [5] could help improve the protocol efficiency, up to 97% if ethernet frames are full. Moreover, a 1-ms-wide bucket can greatly reduce the bandwidth without significantly increasing latency [6].

Early experiments also show that without caching the list of connected clients, the servant-database connection accounts for half to a third of the whole system bandwidth: up to 1.7 MBps for 50 clients and 20 messages per second. The servant-database connection seems to be the main bottleneck and source of extra latency of the whole system.

Finally, our approaches so far have put aside the physics engine. Yet it is a complex and central actor in MMVE architectures. We plan to develop a proof-of-concept of a physics engine for our architecture. The script engine should also be detailed and implemented. Both the physics and script engines have a clock of their own; this may bring additional interesting challenges to our architecture.

References

[1] D. Lake, M. Bowman, and H. Liu, "Distributed scene graph to enable thousands of interacting users in a virtual environment," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, ser. NetGames '10, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1944796.1944815

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

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

[4] Red Dwarf. [Online]. Available: http://www.reddwarfserver.org

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

[6] J. L. Miller and J. Crowcroft, "The near-term feasibility of P2P MMOG's," ser. NetGames '10, 2010.