SEMMO: A Scalable Engine for Massively Multiplayer Online Games

[Demonstration Paper]

Nitin Gupta, Alan Demers, Johannes Gehrke
Cornell University
Ithaca, New York
{niting, ademers, johannes}@cs.cornell.edu

ABSTRACT

We propose to demonstrate SEMMO, a consistency server for MMOs. The key features of SEMMO are its novel distributed consistency protocol and system architecture. The distributed nature of the engine allows the clients to perform all computations locally; the only computation that the central server performs is to determine the serialization order of game actions.

We will demo SEMMO through a game called Manhattan Pals, and show how we can exploit game semantics in order to support large-scale MMOs. In the demo, avatars of the audience will be able to play Manhattan Pals and thus experience various scalability and consistency effects of an MMO.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Concurrency, Distributed Databases*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms

Design, Performance

Keywords

Scalability, Games, Virtual Environments

1. INTRODUCTION

Networked virtual environments (net-VEs) are software systems in which users interact with each other in real-time within some shared virtual environment. *Massively Multiplayer Online Games* (MMOs) are a popular example; these games allow large number of users play together in fictional worlds. Other examples include open virtual worlds like *Second Life* or simulation environments like Microsoft's ESP platform [5]. Other application areas include teaching, distributed design, and military simulations for training and tactical purposes. Virtual environments have become big business, as the MMO *World of Warcraft* has by itself generated up to $1.1 billion dollars of revenue in 2007 [1].

One of the design goals of net-VEs is to create a high degree of immersion. Many of them feature 3D graphics and stereo sound, and they can have extremely interactive environments. But the primary selling point of many net-VEs is the large number of players that they can support. In MMOs like *World of Warcraft* it is already common for groups of players as large as 80 to work cooperatively in a "raid." Other online games like *Habbo Hotel* market themselves as social-networking environments, and need to be able to support large parties or other social events online. Although high-bandwidth, low-latency internet is now becoming ubiquitous, this is not enough to solve the scalability issues that net-VEs encounter as they struggle to support more and more users.

These scalability problems arise in part because of the need to maintain consistency. As in traditional databases, it is very important for a game to be in a consistent state. Violating consistency constraints may just lead to transient visible artifacts with no long-term consequences. However, it can easily cause much more serious problems, like objects being lost or duplicated during a financial transaction. In addition to breaking the realism of the game, consistency issues are a major source of security problems in net-VEs [4]. To address the consistency problem, all net-VEs have a transaction management layer. Because consistency in net-VEs typically has very complex semantics, many of them use commercial databases to handle these transactions. However, as users act in the virtual environment, they send transactions to the net-VE at an extremely high rate. Even the fastest MMOs cannot handle more than about 10 frames per second through their database layer [2].

The transaction layer architecture of most current net-VEs requires that significant parts of their application logic be executed on the server side. As a result, the scalability of an application is strongly related to the computational footprint of a single user. The desire to support more players in complex environments has spawned some research on distributed system architectures for net-VEs; RING is a prototypical architecture [3]. Unfortunately, the scalable examples of these systems have some serious limitations. Often they are limited to narrow classes of transactions, such as those that depend on character visibility. The problem with this approach is that transactions in real net-VEs often interact in complex and subtle ways. For example, suppose we have a fantasy MMO that is designed to support large numbers of players. To make play meaningful, the game designers need to provide players with tools for interacting with large numbers of users. A classic feature for interacting with groups is a "scrying spell" that allows a healer to

Copyright is held by the author/owner(s).
SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.
ACM 978-1-60558-102-6/08/06.

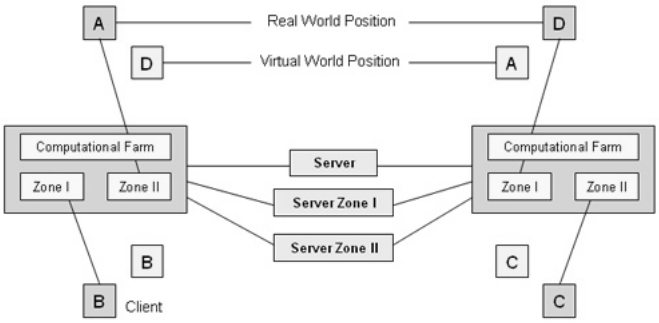

Figure 1: SEMMO System Architecture

Figure 2: The Manhattan Pals Game

identify and heal the most wounded ally in a crowd. During combat, the result of this spell transaction interacts with *all* the other users, as the health of each player is continually changing. The range and nature of such a spell makes character-visibility partitioning useless.

Fortunately, games have a lot of semantic information that can be leveraged to ensure scalable consistency. Games and simulations are essentially high-dimensional databases where the attributes can change For example, health could itself be an attribute that changes as a player is damaged. We can perform range queries on hyperplanes including this health dimension. By examining semantic information such as the maximum damage that an attack transaction can cause, we can predict the ways in which the health attribute can change. We can exploit this semantic information to reduce the number of messages needed to maintain a consistent state among the many distributed clients. This semantic model allows us to easily accommodate many computationally complex net-VEs, allowing for numbers of players comparable to the largest centralized systems.

We will demonstrate a novel consistency server for MMOs developed at Cornell University called SEMMO (**S**calable **E**ngine for **MMO**s). The key features of SEMMO are its novel distributed consistency protocol that takes game semantics into account and its system architecture. The distributed nature of the engine allows the clients to perform all computations locally; the only computation that the central server performs is to achieve consensus on the serialization order of transactions.

We will demo SEMMO through a game called *Manhattan Pals*, and show how we can exploit game semantics such as spatial locality in order to support large-scale MMOs with player and non-player avatars. In the demo, avatars of the audience will be able to play Manhattan Pals. We will first briefly overview SEMMO (Section 2) and then outline the demo (Section 3).

2. SEMMO

In the remainder of this section we will briefly describe the system architecture and the consistency protocol employed by SEMMO.

2.1 System Architecture

The SEMMO system consists of many *computational farms*—a high-bandwidth low-latency network of machines that are geographically dispersed throughout the world. It also consists of a central *server farm*. The virtual world is partitioned into multiple zones. In the server farm, each zone is mapped onto one server, called its *zone server*. One designated server handles fringe effects; we call this server the *fringe server*. The servers in the computational farms are similarly partitioned, and each server in the computational farm is associated with one server in the server farm. The computational farms are geographically located in a manner that minimizes the average network latency for the clients (see Figure 1). The main performance benefit of this architecture is that zoning of the virtual world results in reduced network traffic, whereas a smart geographical positioning of the computational farms reduces the perceived network latency.

We would like to point out that there are many interesting tradeoffs and resulting research problems (for example, positioning and sizing of the computational farms, determination of zones) in this basic part of the architecture, however they are not the focus of the demo.

2.2 Consistency Protocol

We next give a brief overview of our novel distributed consistency protocol that takes the semantics of the actions in the virtual world into account and thus achieves its scalability. For simplicity of discussion, we shall assume that there is one central server and that there are no computational farms.[1]

Each client repeatedly submits *actions* to the central server. An action has an associated read set and write set over the world state, along with the code required to execute the action. On receiving an action A from a client C, the central server computes the set of uncommitted actions that conflict with A (i.e., uncomitted actions whose write sets intersect the read set of A). This conflict set is then extended by transitive closure to the set of all uncommitted actions that might transitively affect the read set of A. The actions in the conflict set, together with values for their read sets, are then sent back to the client C, which evaluates them and sends

[1]A generalization of the consistency protocol follows from the system architecture described in Section 2.1.

the result of its own next action back to the server. The server, upon receiving such a response, records the result and considers the action as committed.

The protocol has two main advantages. First, the server does not perform any costly computation of read and write sets. Second, a client does not (necessarily) evaluate every action in the game world, but only actions that potentially affect any actions of the client. The full details of this protocol are currently under submission, and are outside the scope of this demo description.

3. DEMONSTRATION OUTLINE

We have implemented SEMMO in Java. We will run the live demo (1) on a set of laptops over the local network, and (2) on an EMULab [6] testbed, where we will make remote calls to observe the desired system characteristics. EMULab allows us to simulate different network conditions that will allow us to better demonstrate the effect of our techniques.

We will demo SEMMO using a small, but real game called *Manhattan Pals*, a two-dimensional simulation game in which client avatars — both human player and computer "non-player" avatars — inhabit a two-dimensional virtual Manhattan. The world consists of streets, buildings, and blocks that constrain the movement of participants. There are computer avatars of different complexity. Simple avatars have simple algorithms to change their movement direction by 90 or 180° at various intersections. More complex avatars try to find routes between different tourist attractions in Manhattan (such from as Battery Park to the Empire State Building). The human player avatars can be controlled by the players, who have complete freedom in the choice of their moves.

3.1 Local Demonstration

Game Play. We will first demonstrate how players can interact with each other and with avatars in the virtual world. We have developed a client program for Manhattan Pals (Figure 2), which shows the world state observed by a client at any given time in the game. The visual frontend shows the world state as observed by the player avatar.

Performance. We will vary the following parameters to project their effect on gameplay: (a) the number of non-player avatars in the virtual world; (b) the rate at which non-player avatars make decisions to move; (c) speed of avatars; and (d) the area of impact of their moves. These effects will be experienced by the player avatars who inhabit the virtual world, and the demo audience will be able to experience the differences in performance (and in observed traffic).

Comparison With Previous Work. We will compare the existing RING architecture [3] to SEMMO. We have also developed a frontend for a small data analysis tool that allows us to show various network statistics comparing SEMMO with RING. We will vary various parameters of the underlying network and the game to show the trade-offs on consistency.

3.2 Wide Area Demo

In the second part of the demonstration, we will emulate the SEMMO system architecture over a wide area network. Some machines on the testbed will be designated as servers and others will be organized into computational farms. We will leverage EMULab's capabilities to present a network that closely represents the world wide web.

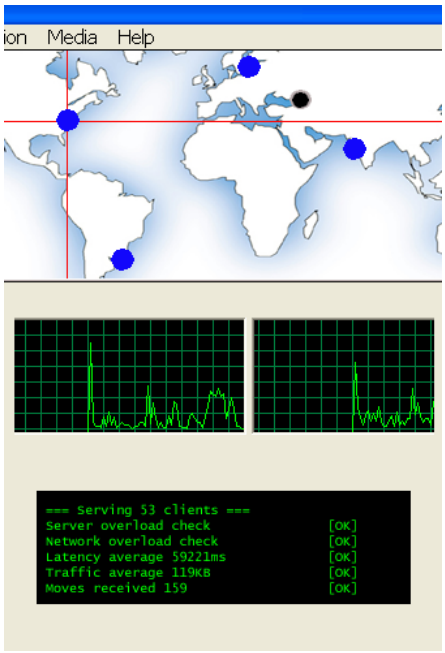

Figure 3: SEMMO Architecture Analyzer

The SEMMO testbed frontend allows the user to choose various geographical locations which will virtually host the computational farms and the server farm on the testbed. The user is also allowed to choose a geographical distribution of the clients from a given set of predefined real world distributions. The Manhattan Pals game will then be simulated on the given network architecture. In particular, we will consider the following scenarios:

Co-located Farms. We will demo the effect on Manhattan Pals when all farms are co-located. We will show the overall network traffic, load on the servers, and average response time observed by the clients as a result of such an architecture.

Widespread Farms. We will demo the effect on Manhattan Pals when the computational farms are geographically distributed according to the distribution of the participants. We show how apparent latency to clients falls sharply as a result of such geographical positioning.

4. CONCLUSIONS

SEMMO is a scalable architecture for MMOs that guarantees consistency while providing massive scalability.

Acknowledgments. This research is based upon work supported by the National Science Foundation under Grant IIS-000492612, by the Air Force under Grant AFOSR FA9550-07-1-0437, and by a Faculty Development Grant from the New York State Foundation for Science, Technology, and Innovation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

5. REFERENCES

[1] S. Carless. http://www.gamasutra.com/php-bin/news_index.php?story=16458.

[2] B. Dalton. Online gaming architecture: Dealing with the real-time data crunch in MMOs. In *Proc. Austin GDC*, 2007.

[3] T. A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *Symposium on Interactive 3D Graphics*, pages 85–92, 209, 1995.

[4] T. Keating. Dupes, speed hacks and black holes: How players cheat in MMOs. In *Proc. Austin GDC*, Austin, TX, September 2007.

[5] Microsoft Corp. http://www.microsoft.com/esp.

[6] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI 2002*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.

[7] W. White, A. Demers, C. Koch, J. Gehrke, and R. Rajagopalan. Scaling games to epic proportions. In *Proc. SIGMOD*, pages 31–42, 2007.