

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

1. INTRODUCTION

A networked virtual environment (net-VE) is a software system in which multiple users interact with each other in real-time, even though those users may be located around the world. These virtual environments provide a shared sense of state (all participants have the illusion of being in the same place), a shared sense of presence (each participant becomes a virtual person, her *avatar*), a shared sense of time (participants see each other's behavior in real time), a way to communicate (for example through gestures, voice, or text), and a way to interact (for example, by giving items to each other or by manipulating items in the environment). These environments aim to provide users with a sense of realism by incorporating realistic 3D graphics and stereo sound, to create an immersive experience.

The collective state of all participants in a net-VE can be represented as a database, known as the *world state*, with each avatar and item mapping to one or more *spatial* database objects. Any interaction with the world becomes a database transaction, i.e. a change in state is equivalent to an update of the database, whereas an observation of state is represented as a read on the database. The ability of users to observe and even modify shared state under a shared presence requires us to *replicate* all or parts of the database between participants.

Massively multiplayer online games (MMOs) represent a very important class of net-VEs. Due to the large number of participants and the interactive nature of the experience, scalability of the underlying computational architecture is crucially important.

Gaming companies today employ many different system models in an attempt to achieve massively scalability. The most popular among these is server-centric *centralized model*, in which the servers perform the computations on behalf of all clients. This model has led to severe scalability problems, in particular for applications that require computationally

expensive tasks (such as realistic game physics) to be carried out centrally. Given the rising cost of energy and server infrastructure, setup and maintenance cost of the infrastructure have become an important factor for game companies.

An alternate model is the client-centric *distributed model*, in which the computations are distributed among clients in order to achieve scalability. Distributing computations among clients can not only reduce load on the central server, but also derive optimal performance from client machines.

While p2p architectures seem to be the natural choice for distributed models, we believe that there is a strong reason to use client-server architectures. Net-VEs are developed and operated by companies that have a vital interest in exerting total control over the game, even if that means investing in server hardware. For games such as Second Life, where players pay real money for game content, the game company has an obligation towards players to provide uninterrupted service. Moreover, it is desirable to have all content stored securely and persistently by a trusted authority.

The key feature of a distributed model is its consistency protocol. Since the computations are performed by the clients, a protocol needs to be established at the clients and the server that ensures consistency, correctness and persistence of data. A typical client-server architecture that uses a distributed model consists of a server cluster to which all clients connect using their *client program*. Client programs implement the game logic, and they initiate and process the moves in the environment. A *move* is a series of atomic actions that update the world state. Typically, each action involves an observation of the world state followed by an update on the state. Since the computations are performed by the clients, a protocol needs to be established at the clients and the server that ensures consistency, correctness and persistence of data. It is the performance of this consistency protocol that makes or breaks the scalability of the distributed model.

In this demonstration, we will show SEMMO,¹ a consistency server for MMOs developed at Cornell University. 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 achieve consensus on the serialization order of transactions.

Since SEMMO is concentrated on the single one issue

¹Scalable Engine for MMOs

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM SIGMOD '08 Vancouver, BC, Canada

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.



Figure 1: The Manhattan Pals Game

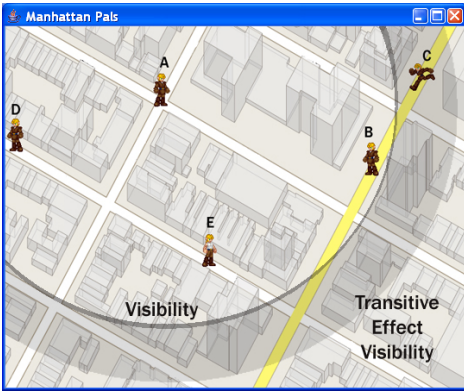


Figure 2: The Impact of Consistency Control

that needs to be handled centrally, SEMMO is light-weight, highly scalable, and it is independent of the game logic. We will demo SEMMO through a game called *Manhattan Pals*, and show how we can exploit trivial game semantics such as spatial locality in order to support large-scale MMOs with player and hundreds of non-player avatars. In the demo, avatars of the audience will be able to play *Manhattan Pals* and thus experience all the effects that we describe above. Let us briefly overview SEMMO (Section 2) and then give an overview of the demo (Section 3).

2. SEMMO

SEMMO’s server-side system architecture consists of a cluster of computers that partition the environment into zones. Servers are arranged hierarchically in accordance with the zoning policy described in Section 2.1 However, the key feature of SEMMO is its distributed consistency protocol that exploits trivial application semantics in order to reduce the number of messages required to maintain consistent state among the many clients distributed across the world. In order to accurately predict the computational requirements of the client and the corresponding network infrastructure, the system provides theoretical bounds on the amount of computation and a measure of bandwidth required at the client machine.

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 architecture consists of many *client farms*—a high-bandwidth low-latency network of machines that can be geographically located anywhere. It also consists of a similar central *server farm*. The introduction of such farms brings around interesting research issues. For example, determining the optimal geographical positioning of client farms in the real world is intuitively non-trivial.

The system partitions the virtual world into multiple zones. In the server farm, each zone is mapped onto one server, called the *zone server*. Another one of the servers is designated to handle fringe effects. We term this server as the *global server*. In the client farms, the system does analogous zone mappings. It then connects the machines across all client farms to the corresponding server at the server farm.

The client farms are geographically located in a manner that minimizes the average network latency to the exhaustive set of clients (Figure 3). The main performance benefit of such an architecture is that zoning of the virtual world results in reduced network traffic, whereas an optimal positioning of the client farms reduces the apparent network latency.

2.2 Consistency Protocol

We next give a brief overview of the distributed consistency protocol. For simplicity of discussion, we shall assume that there is one central server.²

Each client repeatedly submits *moves* to the central server. A move is represented as a read set and a write set over the world state, along with the code required to execute the move. On receiving a move M from a client C , the central server computes the set of uncommitted moves that conflict with M , i.e. there is an overlap in the write set and the read set of the respective moves. This conflict set is then extended to include the transitive closure over all uncommitted moves that conflict with M . This set of moves, along with their respective read sets, is then sent back to the client C , which evaluates them and sends the result back to the server. The server, upon receiving such a response, records the result and considers the move as committed. The advantage of this model is that a client does not (necessarily) evaluate every move, but only those that affect it. To deterministically evaluate the effects of a move, a client needs to be informed about moves in a modified *transitive closure* of its effects; we can visualize this as shown in Figure 2. The full details of this protocol are under submission at SIGMOD 2008, and they are outside the scope of this demo description.

3. DEMONSTRATION OUTLINE

We have implemented SEMMO in Java 5.0. We will run the live demo (1) on a set of laptops over the local network, and (2) on an EMULab [?] 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 which consists of client avatars — both human player and com-

²A generalization of the consistency protocol follows trivially from the system architecture described in Section 2.1.

puter “non-player” avatars — that 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 sights on Manhattan (such from as Battery Park to the Empire State Building). The human players can be controlled by human players who have complete freedom in the choice of their moves. The game creates a virtual environment with some of the basic functionality outlined in the introduction (see Figure 1).

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).

Consistency. We will demonstrate how consistency affects throughput in Manhattan Pals. We also show how an increase in consistency requirements results in severe scalability and bandwidth problems for the RING architecture compared to SEMMO. We also show graphically the part of the virtual world that a client needs to be aware of according to our protocol 2.

We have also developed a frontend for a network & data analyzer that for a simulation in fast-forward mode allows us to plot various statistics comparing SEMMO with RING, a consistency violating architecture [?]. We will vary various parameters of the underlying network and the game to show the tradeoffs 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 client farms. We will leverage EMULab’s interface to present a network that closely represents the world wide web.

The SEMMO frontend to its system architecture allows the user to choose various geographical locations which will virtually host the client 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.

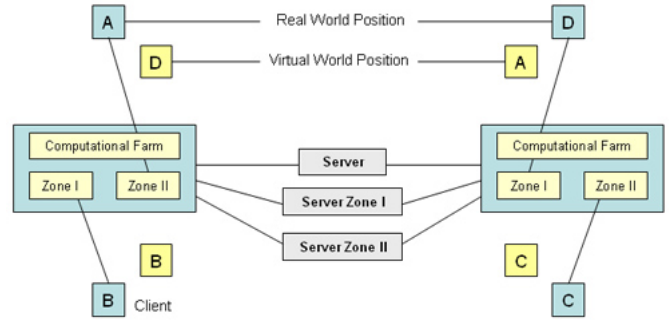


Figure 3: SEMMO System Architecture

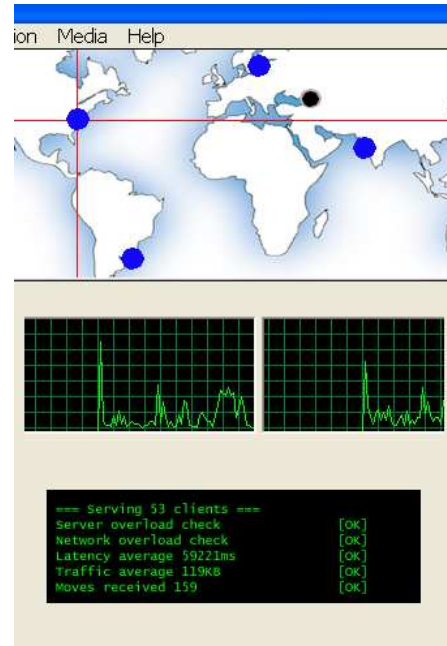


Figure 4: SEMMO Architecture Analyzer

Widespread Farms. We will demo the effect on Manhattan Pals when the client 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.

Conclusions. SEMMO is to the best of our knowledge the first scalable architecture for MMOs that guarantees consistency while providing massive scalability. We believe a demo of its capabilities will showcase this exciting research area to the database community.