Scalability for Virtual Worlds

Nitin Gupta #, Alan Demers #, Johannes Gehrke #, Philipp Unterbrunner *, Walker White #

#Computer Science, Cornell University, Ithaca, NY *Systems Group, ETH Zurich, Switzerland
{niting,ademers,johannes,wmwhite}@cs.cornell.edu, philppu@inf.ethz.ch

I. Introduction

Networked virtual environments (net-VE) are software systems in which users interact with each other in real-time within some shared virtual environment. The most popular example are *Massively Multiplayer Online Games* (MMOs), but they also include *virtual worlds* such as *Second Life*, or simulation environments like Microsoft's ESP platform [1]. Net-VEs have applications in teaching, distributed design, and military simulations for training and tactical purposes.

A major selling point for net-VES is the number of concurrent users. In MMOs like *World of Warcraft* it is already popular for groups of up to 80 players to work cooperatively in a "raid". In Net-VEs focused on social networking, like *Second Life* or *Habbo Hotel*, online gatherings and parties are very popular. The greatest challeng in supporting so many players is maintaining consistency. Consistency violations degrade the realism of the virtual world, and are a major source of security problems in net-VEs [2].

To maintain consistency, most net-VEs have a transaction management layer using a commercial database. However, this does not scale well. Users send transactions at an extremely high rate; even the fastest MMOs cannot handle more than 10 frames worth of transactions per second [3]. In addition, much of the application logic must be executed on the server side, so the scalability of a net-VE is depends strongly on the computational footprint of a single user.

Fortunately, net-VEs are essentially high-dimensional databases whose attributes change in predictable ways [4]. For example, in a fantasy MMO game, health is an attribute that changes as a player is damaged, and this damage is caused by transactions specified in advance. Using semantic analysis of these transactions, we can reduce the number of messages needed to maintain consistency.

In this paper, we propose a distributed model for massive scalability in net-VEs. Our model inherits concepts from distributed databases, where the application logic and transaction processing take place at the client machine. The key feature of our model is its novel transaction model, which exploits application semantics to reduce the number of messages needed to maintain consistency. Our proposed model imposes no major restrictions on the interaction between participants located in different parts of the world, and so can be easily adapted to a wide range of net-VEs.

II. Networked Virtual Environments

To achieve realism, every machine needs to share a consistent view of the virtual environment. Net-VEs typically store this *world state* in a database [5]. Any interaction in the world can be thought of as a database transaction. However, because of throughput problems with commercial databases, most net-VEs use commercial databases only for periodic checkpoints, implementing their own lightweight transaction layer in front of the database [3].

Due to space constraints, we will focus only on client-server net-VEs. A client-server net-VE architecture consists of a server cluster to which all clients connect. The clients run identical net-VE*client programs*, which contain the virtual world logic. Clients initiate and process *actions* in the environment. An action is a sequence of atomic operations typically consisting of an observation of the world state followed by an update of the state. For simplicity, we assume each action consists of exactly one atomic operation. Processing actions in the client program may raise security issues; these have been addressed in prior research [13], [14], [15].

The key component of a client-server net-VE is the *consistency protocol* between the clients and the serve, which ensures consistency and durability of data. *Lock Based*, *Timestamp Based* [16] and *Object Ownership* [17] are three popular classes of such protocols. Each of these can exhibit slow response time to the client. For example, Sun's Project Darkstar [18], which uses a timestamp based protocol, requires an extended response time to enforce consistency. Another problem is that the consistency resolution is object based, while many consistency problems in net-VEs are semantic. The virtual world designer is forced to map every consistency issue to an object access, which is not always easy.

III. Action Based Protocols

The techniques discussed in the previous section handle consistency at the object level. In this section, we describe a novel class of protocols that we call *action based protocols* because they check consistency at the level of actions rather than objects.

A. The Basic Algorithms

In our action based protocols, the messages passed between the clients and the server primarily consist of *actions*, as opposed to objects. The state of the virtual world is a database of objects, the *world state*. Each client program maintains two versions of the world state: an optimistic version ζ_{CO} and a stable version ζ_{CS}. To perform an action a, a client first applies a to ζ_{CO}, and also sends a to the server to be serialized. Concurrently, the client is receiving from the server a serialized stream of the actions originating at *all* clients,

1084-4627/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDE.2009.228

Algorithm 1: Client-Side Protocol

1 The client maintains a queue

$$\mathcal{Q} = [\langle a_1, v_1\rangle, \ldots, \langle a_k, v_k\rangle]$$

where each a_i is a locally generated action that has not yet been received back from the server, and v_i is the result of applying a_i to ζ_{CO} as described below.

2 Whenever the client creates an action a, the action is first executed on ζ_{CO} producing a result v. We call this the optimistic evaluation of a. The pair $\langle a, v\rangle$ is then added to $\mathcal{Q}$, and the action a is sent to the server.

3 Assume that the client receives an action b from the server. There are two possible cases:

4 (Action b originated at some other client): Action b is applied to ζ_{CS}. Each write $x \leftarrow v$ performed by b is also performed on ζ_{CO} if (and only if) $x \notin WS(\mathcal{Q})$. (This has the effect of updating items in the state that are not awaiting permanent values from the server).

5 (Action $b = a_1$): Action a_1 is applied to ζ_{CS} producing result u. If $u = v_1$, indicating the new evaluation of a_1 agrees with its optimistic evaluation, the entry $\langle a_1, v_1\rangle$ is removed from the head of $\mathcal{Q}$. Otherwise, ζ_{CO} is reconciled with ζ_{CS} using Algorithm 3.

Algorithm 2: Server-Side Protocol

1 The server maintains a global queue of actions. For each client C, the server maintains the index pos_C of the action in the queue that was last sent to C. At the start of the protocol, $pos_C = 0$ for all clients C.

2 When the server receives an action a from client C (Step 2 in the client-side protocol), it performs two steps:

3 (a) It timestamps a and puts it into the queue, assigning a a unique order number $pos(a)$ that is a's position in the queue.

4 (b) The server returns to C all actions between positions pos_C and $pos(a)$, and it sets $pos_C = pos(a)$.

and applying them in order to ζ_{CS}. The results of applying locally originated actions to ζ_{CO} and ζ_{CS} are compared, and disagreements are reconciled if necessary. Pseudocode is presented as Algorithms 1, 2 and 3.

Note that the only function the server provides is to timestamp and serialize the actions of the clients. The timestamp, together with the positions of actions on the queue at the server, establishes virtual synchrony between the server and the clients [12].

The reconciliation procedure in our protocol, Algorithm 3, prevents the optimistic state from diverging too far from the stable state, by rolling back and re-applying optimistic actions when an actual conflict is discovered. This approach is similar to that used in Bayou [19], and like Bayou we assume that actions contain code to check for conflicts: when it is re-applied, an action either computes appropriate new result values or else detects a fatal conflict and behaves as a no-op to simulate aborting.

Correctness of our protocol is easy to establish: Each client executes every action that originates anywhere in the system, *in the same order* on its stable version of the world ζ_{CS}.

Our action-based protocol has two advantages. First, it guarantees response in one round trip while allowing any kind of interaction in the virtual environment. Second, it frees the central server from dependence on the game logic.

Algorithm 3: Reconciliation Protocol

Require: $\mathcal{Q} = [\langle a_1, v_1\rangle, \ldots, \langle a_k, v_k\rangle]$ is the results of optimistic evaluation of locally generated actions.

$\zeta_{CO}(WS(\mathcal{Q})) \leftarrow \zeta_{CS}(WS(\mathcal{Q}))$

$\mathcal{Q} \leftarrow []$

for $(j = 1; j <= k; j++)$ **do**

  apply a_i to ζ_{CO} producing result v

  insert $\langle a_i, v\rangle$ into $\mathcal{Q}$

However, a major drawback of this protocol is that every client sees and executes all actions for the entire world, resulting in high computational load at the clients as well as substantial bandwidth requirements. Thus this first protocol, while ensuring good response time and achieving consistency, has very limited scalability. To improve scalability we exploit application semantics, as described next.

B. Using Application Semantics

In the realm of object based protocols, many optimizations have been proposed to reduce the number of messages [20], [21]. Most of these are variants of *area-of-interest* paradigm [22], [23], in which the server restricts the set of messages sent to a client by some syntactic constraint, for example, the visibility of an avatar in the virtual world. It is natural to consider applying the scalability solutions proposed in these systems to our action-based protocols. However, we argue that this approach has problems that prevent it from being a general solution to the scalability problem.

The first problem is that restricted visibility applies only to movement-like actions and does not generalize well to arbitrary actions. For example, the RING architecture requires that the designer create an obstruction layer representing the objects blocking visibility. This obstruction layer is used to partition the database replicas [20]. If the game designer wants to base actions on other senses such as sound or scent, she must create a separate obstruction layer for each new sense. In some cases, such as casting a spell in a fantasy game, there may be no notion of "proximity" or "line of sight," hence no useful obstruction information at all.

The use of restricted visibility has a deeper, subtle problem: Characters can interact *transitively* even if they cannot see one another directly. Since the transitive closure of the visibility relation is infeasibly large, current proposals use only direct visibility, and thus fail to handle transitive interactions correctly.

C. The Incomplete World Model

Let us examine an action from a database perspective. An action a consists of a read set $RS(a)$, a write set $WS(a)$ and the code that needs to be executed to compute values for $WS(a)$ given values for $RS(a)$. We assume $RS(a) \supseteq WS(a)$. This allows us to drop the distinction between read sets and write sets and focus on intersecting read sets in our discussion and protocols. Our algorithms occasionally use special *blind*

Algorithm 4: Incomplete World Client-Side Protocol

1 ...
2 ...
3 ...
4 (Action b originated at some other client, *or is a blind write created by the server*): ...
5 ... Finally, a *completion message* $\langle a_i, u \rangle$ is sent to the server.

Algorithm 5: Incomplete World Server-Side Protocol

1 The server maintains the *authoritative state* ζ_S. It also maintains a global queue of ordered actions. For each action a in the queue it maintains the set $sent(a)$ of clients to which the action has been sent.
For any i, let $\zeta_S(i)$ be the state of the virtual world at the server after applying the effects of actions $a_1 \dots a_i$. Then at time t, the server holds: (i) $\zeta_S(j)$ for the least j such that no response for a_{j+1} has yet been received, and (ii) $a_{j+1} \dots a_n$.
2 When the server receives an action a from client C (Step 2 in the client-side protocol), it performs two steps:
3 (a) It timestamps a and puts it into the queue, assigning a a unique order number $pos(a)$ that is a's position in the queue. It also sets $sent(a) \leftarrow \emptyset$
4 (b) It computes a reply to a using Algorithm 6.
5 When a completion message arrives at the server for a_i (Step 5 in the client-side protocol), the server holds it until $\zeta_S(i-1)$ is available. It then installs the values into ζ_S, resulting in $\zeta_S(i)$, and discards a_i from the action queue.

write actions: we denote by $a = W(S, v)$ an action that unconditionally stores the values v into the object set S (assuming compatibility of S and v). Clearly $WS(a) = S$, and by convention $RS(a) = S$ as well. We can now change our protocols as shown in Algorithm 4 (which is just two small changes to Algorithm 1) and Algorithms 5 and 6.

The advantage of this model is that a client does not (necessarily) evaluate every action, only those that affect it, thus saving execution time at the clients as well as network bandwidth. To achieve this goal, we augment the client protocol to return a *completion message* when the stable result of an action is produced. The server uses these messages to construct ζ_S, an authoritative stable world state. The server performs analysis of read and write sets (Algorithm 6) to determine independently for each client which additional actions must be sent for evaluation because they (transitively) affect the client's submitted actions.

An interesting aspect of this model is that it can be made tolerant of client failures at a reasonable cost in network bandwidth, by letting each client send completion messages for *every* action it applies, not just its own. With this, the only case in which the server does not receive a response to some action is when all clients that evaluate that action have failed.

IV. Experiments

We built a system implementing the action based protocol in Java 5.0 and conducted experimental studies to evaluate its performance. We call our implementation *SEVE*, for Scalable Engine for Virtual Environments. Our experimental evaluation is based on a synthetic workload that stresses the consistency issues in MMOs.

Algorithm 6: Transitive Closure(A)

Require: $a_i, \dots, a_n$ is the action queue
Require: a_{n+1} has just arrived from client C
Require: $+$ denotes prepending an action to a sequence
$A \leftarrow \{a_{n+1}\}$
$S \leftarrow RS(a_{n+1})$
for $(j = n; j > i; j = j - 1)$ **do**
  if $WS(a_j) \cap S \neq \emptyset$ **then**
    if $C \in sent(a_j)$ **then**
      $S \leftarrow S \setminus WS(a_j)$
    else
      $S \leftarrow S \cup RS(a_j)$
      $A \leftarrow a_j + A$
      $sent(a_j) \leftarrow sent(a_j) \cup \{C\}$
$A \leftarrow W(S, \zeta_S(S)) + A$
return A

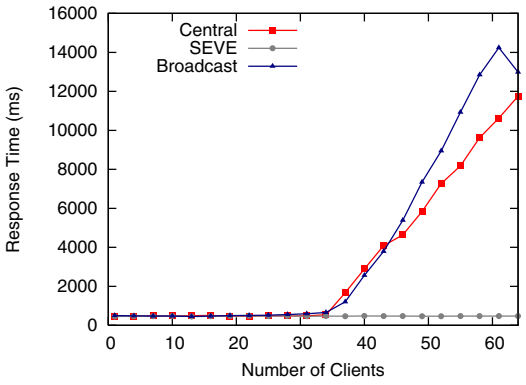

Fig. 1. Scalability of SEVE vs. Central architecture

A. Experimental Setup

All performance results were obtained by running the virtual world on an EMULab [24] testbed consisting of 65 machines—64 clients and 1 server. Each client machine was running other programs, such as a desktop manager, in the background. The average latency between machines was 238ms.

B. Performance Evaluation

We performed multiple experiments. First, we evaluated the scalability-complexity tradeoff in (a) a centralized model (Central)—to represent Second Life and WoW, the state of the art in online games; (b) a broadcast model (Broadcast)—representing distributed simulations such as HLA, NPSNET and SIMNET; and (c) our action based distributed model (SEVE). Second, we explored the bandwidth requirements of the three models.

Figure 1 plots the response time observed by clients against the number of clients. Both the centralized architecture and the broadcast model break down at about 30-32 clients. In contrast, SEVE's response time remains perfectly stable as the number of clients increases.

Figure 2 plots the response time observed by the clients against the time required evaluate a single move. The number of clients was fixed at 25. The centralized model and broadcast model performed well for moves requiring less than 10ms.

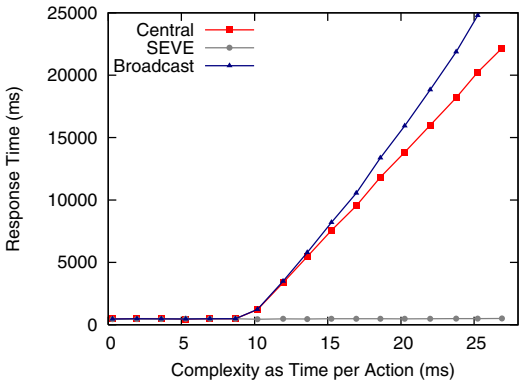

Fig. 2. Response Time vs. Action Complexity

However, as the complexity increased, the response time increased dramatically. Again, the response time for SEVE remained unaffected.

In summary, our experiments show that our architecture achieves scalability while preserving strong consistency.

V. Related Work

Reality Built For Two [25] and MR Toolkit [26] are two net-VEs that maintain consistent state among N workstations by sending a point-to-point message to each of the workstations for every state change. This approach yields $O(N^2)$ update messages per simulation step, and does not scale. NPSNET [27] follows a basic object based broadcast model. It broadcasts messages to all workstations at once, yielding $O(N)$ update requests for N workstations. However, the computational requirement at each client is the same as MR Toolkit. RING [20] and DIVE [21] handle message filtering by sending all updates to the central server. The server tracks the current location of each entity, and it can determine which users might be interested in a particular update. This approach is close to our model, but in both these systems the server forwards updates only to users who can "see" the entity, leading to inconsistency (cf. Section III-B). Wiesmann et al. [28], [29] evaluate replication techniques based on broadcasting events in total order. Our algorithm can be understood as a fast-paced instance of 2-tier replication [30], which in turn builds on multi-version serializability theory [31].

VI. Conclusions

In this paper we argued that data management problems lie at the core of networked virtual environments. We identified an interesting concurrency problem and proposed a novel practical solution based on taking semantics into account. We believe that we have just scratched the surface of this new area, and that virtual worlds as well as other virtual networked environments will benefit from solutions from the database community for years to come.

References

[1] Microsoft Corp, "http://www.microsoft.com/esp."
[2] T. Keating, "Dupes, speed hacks and black holes: How players cheat in MMOs," in *Proc. Austin GDC*, 2007.
[3] B. Dalton, "Online gaming architecture: Dealing with the real-time data crunch in mmos," in *Proc. Austin GDC*, 2007.
[4] W. White, A. Demers, C. Koch, J. Gehrke, and R. Rajagopalan, "Scaling games to epic proportions," in *Proc. SIGMOD*, pp. 31–42, 2007.
[5] R. Bartle, *Designing Virtual Worlds*. New Riders Games, 2003.
[6] I. Kazem, D. T. Ahmed, and S. Shirmohammadi, "A visibility-driven approach to managing interest in distributed simulations with dynamic load balancing," in *Proc. DS-RT*, pp. 31–38, 2007.
[7] Wikipedia, "Instance dungeon." [Online]. Available: http://en.wikipedia.org/wiki/Instance_dungeons
[8] R. R. Koster, "From instancing to worldy games." [Online]. Available: Raph Koster's Blog: http://www.raphkoster.com
[9] Tobold, "Servers and critical mass." [Online]. Available: Tobold's MMORPG Blog: http://tobolds.blogspot.com
[10] A. Taylor, "The problem with world of warcraft." [Online]. Available: Wonderland Blog: http://www.wonderlandblog.com
[11] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.
[12] K. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems," in *Proc. SOSP*, pp. 123–138, 1987.
[13] A.-R. Sadeghi and C. Stüble, "Taming "trusted platforms" by operating system design," in *WISA*, 2003, pp. 286–302.
[14] A.-R. Sadeghi and C. Stüble, "Property-based attestation for computing platforms: caring about properties, not mechanisms," in *Proc. NSPW*, pp. 67–77, 2004.
[15] P. Kabus, W. W. Terpstra, M. Cilia, and A. P. Buchmann, "Addressing cheating in distributed mmogs," in *Proc. NetGames*, pages 1–6, 2005.
[16] M. K. Sinha, P. D. Nandikar, and S. L. Mehndiratta, "Timestamp based certification schemes for transactions in distributed database systems," in *Proc. SIGMOD*, pp. 402–411, 1985.
[17] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: a distributed architecture for online multiplayer games," in *Proc. NSDI*, pp. 12–12, 2006.
[18] Sun Microsystems, "Project Darkstar." [Online]. Available: http://www.projectdarkstar.com
[19] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer, "Bayou: replicated database services for world-wide applications," in *Proc. SIGOPS European workshop*, pp. 275–280, 1996.
[20] T. A. Funkhouser, "RING: A client-server system for multi-user virtual environments," in *Proc. SI3D*, pp. 85–92, 209, 1995.
[21] O. Hagsand, R. Lea, and M. Stenius, "Using spatial techniques to decrease message passing in a distributed VE system," in *Proc. VRML, pp. 7–ff, 1997.*
[22] S. Han, M. Lim, and D. Lee, "Scalable interest management using interest group based filtering for large networked virtual environments," in *Proc. VRST*, pp. 103–108, 2000.
[23] G. Morgan, F. Lu, and K. Storey, "Interest management middleware for networked games," in *Proc. I3D*, pp. 57–64, 2005.
[24] 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 *Proc. OSDI*, pp. 255–270, 2002. USENIX Association.
[25] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel, "Reality built for two: a virtual reality tool," in *Proc. SI3D pp. 35–36, 1990.*
[26] C. Shaw and M. Green, "The MR toolkit peers package and experiment," in *VR*, pp. 463–469, 1993.
[27] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz, "NPSNET: A network software architecture for large-scale virtual environment," in *Presence*, vol. 3, no. 4, pp. 265–287, 1994.
[28] M. Wiesmann, "Comparison of database replication techniques based on total order broadcast," in *IEEE TKDE*, pp. 551–566, 2005.
[29] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems," in *Proc. ICDCS*, p. 464, 2000.
[30] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proc. SIGMOD*, pp. 173–182, 1996.
[31] P. A. Bernstein and N. Goodman, "Concurrency control algorithms for multiversion database systems," in *Proc. PODC*, pp. 209–215, 1982.