Scalability for Virtual Worlds

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

[#]*Computer Science, Cornell University*
Ithaca, NY

[*]*Computer Science, ETH Zurich*
Zurich, Switzerland

{niting,ademers,johannes,wmwhite}@cs.cornell.edu, philppu@inf.ethz.ch

***Abstract*— Networked virtual environments (net-VEs) are the next wave of digital entertainment, with Massively Multiplayer Online Games (MMOs) a very popular instance. Current MMO architectures are server-centric in that all game logic is executed at the servers of the company hosting the game. This architecture has lead to severe scalability problems, in particular since MMOs require realistic graphics and game physics – computationally expensive tasks that are currently computed centrally.**

We propose a distributed *action based* protocol for net-VEs that pushes most computation to the computers of the players and thereby achieves massive scalability. The key feature of our proposal is a novel distributed consistency model that allows us to explore the tradeoff between scalability, computational complexity at the server, and consistency. We investigate our model both theoretically and through a comprehensive experimental evaluation.

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. *Massively Multiplayer Online Games* (MMOs), and more specifically, *virtual worlds* such as *Second Life* are a popular example; these games allow large number of users to play together in fictional digital worlds. Other examples include simulation environments like Microsoft's ESP platform [1]. 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* by itself generated up to $1.1 billion dollars of revenue in 2007 [2].

Virtual worlds are typically designed to create a very high degree of immersion. Many feature 3D graphics and stereo sound, and have extremely interactive environments. But the primary selling point of many virtual worlds is the large number of players that they can support. In MMOs like *World of Warcraft* it is already popular for groups of up to 80 players to work cooperatively in a "raid". Other online virtual worlds like *Habbo Hotel* market themselves as social-networking environments, and must support large parties or other social events online. While high-bandwidth, low-latency internet is now becoming ubiquitous, this is not enough to solve the scalability issues that net-VEs are beginning to encounter.

These scalability problems arise in part because of the need to maintain consistency between all the players. In the best case, inconsistency may just lead to transient visible artifacts with no long-term consequences. However, in practice, it can easily cause much more serious problems, like objects being lost or duplicated during a financial transaction. In addition to degrading the realism of the virtual world, consistency violations are a major source of security problems in net-VEs [3]. To maintain consistency, all net-VEs have a transaction management layer that employs a commercial database.

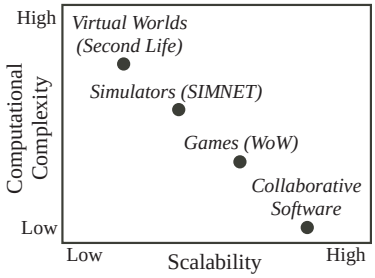

Fig. 1. Scalability versus Complexity

However, the transaction layer also introduces severe scalability problems. First, as users move about 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 [4] through their database transaction layer. Second, 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. Figure 1 illustrates this observation for a sample of todays net-VEs. Collaborative software such as Wikipedia is highly scalable because user actions involve only simple computations. MMO Games with a static environment such as *World of Warcraft* require comparatively more computational resources, leading to a drop in scalability. Simulators, particularly military simulators such as SIMNET, are even more "real" than virtual worlds, in that users can interact with the virtual environment (e.g., destroy buildings); the result is even less scalability. Finally, user-designed virtual worlds such as *Second Life* allow objects to be created, modeled, and scripted by the users at run-time. This flexibility comes with high computational complexity; for example, the resulting scalability of *Second Life* is on the order of at most 25-30 users per server. If the player-to-server ratio of collaborative software were possible in a net-VE with the flexibility and degree of immersion of virtual worlds, this would allow for a user experience beyond the reach of current systems.

The desire to support more players in complex environments has spawned some research on distributed system architectures for net-VEs [5], [6], [7], [8]. Unfortunately, as we discuss in Section II, these systems do not scale. They are often 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 beyond visibility. For example, suppose we have a fantasy MMO designed to support a large number of players. A classic feature for such a game is a "scrying spell" that allows a healer to 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, virtual worlds have a lot of semantic information that can be leveraged to ensure scalable consistency. Virtual worlds and simulations are essentially high-dimensional databases where the attributes can change only in predictable ways [9]. For example, in a fantasy MMO game, health is itself an attribute that changes as a player is damaged. 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, and exploit this semantic information to reduce the number of messages needed to maintain a consistent state among the many distributed clients.

In this paper, we propose a distributed model for networked virtual environments that achieves massive scalability. Our model inherits concepts from distributed databases, where the application logic and transaction processing take place at the client machine, thereby relieving the server from much computation. 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 among the clients. Our proposed model imposes no major restrictions on the interaction between participants located in different parts of the world. As a result, we show how computationally expensive net-VEs can easily be accommodated in our model and can be scaled to a massive number of participants.

Outline of the Paper

In Section II, we overview existing architectures of net-VEs and virtual worlds. We also connect net-VEs and databases, and discuss scalability limitations of today's net-VEs. Our paper then continues with the following contributions.

- In Section III, we introduce a new protocol to increase the scalability of net-VEs. We use application semantics to provide theoretical bounds that show the scalability of our approach.
- In Section IV, we present several techniques that leverage spatial properties of net-VEs to optimize our protocols.
- In Section V, we present an experimental evaluation using both simulation and real experiments demonstrating the effectiveness of our new protocol.

We discuss related work in Section VI and conclude in Section VII.

II. Networked Virtual Environments

As discussed in the previous section, consistency is important to net-VEs. In order to be realistic, everyone needs to share a single view of the virtual environment, the *world state*. Persistent net-VEs typically store the world state in a database [10]. Any interaction in the world can be thought of as a database transaction: making an observation is a database query about the state of the world, and a change in state is a database update. However, because of throughput problems with commercial databases, most net-VEs use commercial databases only to commit and read at periodic checkpoints. For real-time interactions, they generally implement their own in-memory transaction layer in front of the database. This design decision is not because database transactions are unsuited to the task; rather, it is because existing commercial databases are not optimized for the type of in-memory processing that net-VEs need for real-time performance [4].

A. Net-VE Architectures

1) Centralized VEs: MMO companies typically use an architecture with multiple central servers to achieve scalability. The most widely used techniques to achieve scalability beyond a single server are the following:

Zoning. Zoning refers to the technique of geographically partitioning ("tiling") the virtual environment into areas small enough for a single server to handle. Typically, players within a zone form an event broadcast group and are assigned to the same server. A single server machine may host multiple zones to improve resource utilization. Some MMOs allow players to move between zones–this requires a model that allows overlapping of zones. However, great complications arise from attempts to overlap zones [11].

Sharding. Zoning works well to about a few dozen servers, which translates into a few thousand players for most virtual worlds. At this point, scaling effects force MMO companies to instantiate completely separate world instances called *shards*. Sharding is also used because players are potentially spread across the globe; a player from India may not be willing to accept a noticeable 300ms delay in server interactions for the purpose of playing with people from the US. So rather than hosting a single virtual environment for the whole user base, companies serve dozens of wholly disjoint instances of the same world at strategic locations across the globe.

Instancing. Unlike sharding, *instancing* is confined to small partitions of the virtual environment. An instance is essentially a private zone into which no players may enter except those that originally spun off the instance. In *World of Warcraft*, instancing is used heavily for dungeons that are intended to be personal experiences [12]. In general, instancing is often employed for game design reasons; it is not intended to be a true scalability solution, as it severely limits player interaction.

All these solutions split the user base in some way, degrading the "massive" multiplayer experience [13]. For example, sharding and instancing prevent large groups of users from working together by design, while zones collapse if too many users crowd into a zone all at once [14]. Users often

have difficulty finding their real life friends in such MMOs. Some virtual worlds even require the users to pay if they want to play with someone of their choice [15]. Therefore, MMO companies are still struggling to meet the scalability requirements demanded by their user base.

2) Distributed VEs: An alternative to multiple central servers is the *distributed model* [10], in which computation is distributed among clients in order to achieve scalability. Distributing computation between clients has the potential not only to reduce load on the central server, but also to leverage capabilities of the client machines.

While P2P architectures seem to be the natural choice for distributed models, there are both technical and non-technical reasons to choose a client-server architecture. First, *strongly consistent* P2P architectures do not scale because they use protocols such as Paxos [16] or Virtual Synchrony [17] to enforce a consistent total order of events across all participants. Much complication and inefficiency in these protocols arises from the attempt to consider all hosts in the P2P system as first-class, but unreliable, citizens [18].

However, there is an even stronger, non-technical reason to rely on a client-server architecture. Net-VEs are developed and operated by companies that have a vital interest in exerting total control over the virtual world, even if that means investing in server hardware. In many virtual worlds, players pay real money both to participate and for game content; hence the MMO company has an obligation to provide uninterrupted service. Additionally, cheating is a major concern for action-oriented MMOs, and it is much easier to prevent in a client-server architecture [19]. For these and other reasons, companies desire to have all content stored securely and persistently by a trusted authority.

Overall, the client-server distributed model strikes a balance between preserving the interests of the MMO companies in exerting control, scalability of the system, and alleviating the problems of no centralized control compared to a P2P architecture. Thus, we will adopt it in the remainder of the paper. Extensions to a hybrid architecture that strives a balance between P2P and client-server are an interesting direction for future work (see Section VII).

B. Client-Server Net-VEs

A client-server net-VE architecture consists of a server cluster to which all clients connect. Without loss of generality, the clients run identical net-VE software, which we refer to as *client programs*. The client program contains the actual virtual world logic. Clients initiate and process *actions* in the environment. An action is a sequence of atomic operations that updates the world state. Typically, each action involves first an observation of the world state followed by an update of the state. In this paper, we assume that each action consists of exactly one atomic operation. Though processing actions in the client program may raise security issues, a lot of prior research already exists for developing non-hackable clients [20], [21]. As an added security measure, the servers can also log MMO statistics to detect any cheating or security threat [19].

The key component of a client-server net-VE is its consistency protocol. Since the computation is performed by the clients, a protocol needs to be established between the clients and the server that ensures consistency and durability of data. The following are three popular classes of protocols:

Lock Based Protocols. An well-known family of protocols is based on distributed locking. In order to process a transaction, a client must acquire global locks on the objects read and written by the transaction. This can be implemented by having all clients in the system agree on granting a lock request, or by managing locks at the server-side. Typically, a client contacts the server for a lock. The server multicasts this request to all clients, which then respond with an acknowledgement granting the lock request. Any other conflicting transaction requesting the same lock is blocked. The server then communicates to the client the status of its lock requests. If it obtained all the necessary locks, the client executes the transaction on its local state and transmits the effect of the transaction to the server. The server then transmits this effect to all other clients who update their own local states and proceed to the next transaction. Sun's Project Darkstar is an example of a net-VE architecture using a lock based protocol [22].

There are two major problems with using distributed locking in net-VEs. The first problem is that that the minimum time required by a client to proceed to the next conflicting transaction is twice the round trip time between the client and the server; this is often unacceptable performance for net-VEs. The second problem is that the consistence resolution is object based, while many consistency problems in net-VEs are semantic. The virtual world designer is forced to map every single consistency issue in the world to an object access, which is not always easy to do.

Timestamp Based Protocols. The well-known alternative to locking in a distributed database is optimistic concurrency control based on timestamps. Here, we associate a version with every object, and a timestamp with every transaction; the timestamp can be assigned by the server. Clients optimistically execute tentative actions against their local, possibly stale versions of objects. The server integrates the local, transactional histories submitted by clients into a global multiversion history. It can then choose to make decisions on the success of transactions based on multiversion serializability, or broadcast the global history, in which case clients can use certification algorithms to reach consistent commit and abort decisions [23].

If the server makes commit and abort decisions, then it must either be pessimistic, or it must understand game-specific logic and perform possibly expensive operations in order to resolve conflicts. For example, any change in the read set of a transaction, such as some player moving, would potentially cause the transaction to abort. In order to neglect irrelevant changes, the server must implement a significant part of the application logic that specifies what combination of movements are valid.

If the server broadcasts the global history to clients it uses at least two phases, and therefore at least twice the round trip time between the client and the server [23]. There has been

very little research on optimizations of such a protocol to make them suitable for MMOs. Instead, designers prefer protocols based on object ownership that we will discuss next.

Object Ownership. Object ownership differs from lock-based protocols in that each object is owned and managed by exactly one client, known as the object owner. Other clients are allowed to cache a version of the object, but are not allowed to make modifications to its state. RING [6], Cyberwalk [24], and WAVES [25] are three popular systems using such a protocol. Although this protocol is highly scalable, it does not allow for any kind of object contention in the environment. If two clients want to change the state of the same object, only the client owning the object is permitted do so. The server is responsible for ensuring a fairness in ownership, which is often based on semantics of the virtual world. In order to allow object contention in such a protocol, applications are either degraded to a lower level of consistency, or are forced to employ timestamp-based serializability [26], resulting in unacceptable response time for net-VEs.

III. Action Based Protocols

All 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 (i.e. functions to update the game state) rather than at the level of objects. We then propose models based on such protocols that are practical and highly scalable. We use application semantics to provide theoretical bounds that formally prove the scalability of our approach.

To make the discussion more succinct and relevant to current MMOs, we assume that the virtual world follows the standard model of a discrete simulation engine, where the world state changes only at regular time intervals, the *simulation ticks* [9]. We denote the non-zero time interval between two consecutive ticks by τ.

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, 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 2.

Note that the only function the server provides is to timestamp and serialize the actions of the clients. This virtual timestamp, together with the positions of actions on the queue at the server, establishes virtual synchrony between the server and the clients [17].

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 $pos_C = pos(a)$.

The reconciliation procedure in our protocol, Algorithm 3, is designed to prevent 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 [27], 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 it detects a fatal conflict and behaves as a no-op to simulate aborting.

Correctness of our protocol is easy to establish since each client executes every action that originates anywhere in the system, *in the same order* on its stable version of the world ζ_{CS} as enforced by the timestamping and ordering of actions on the server.

Our action-based protocol has two advantages. First, it guarantees response in one round trip while allowing any kind of interaction, including object contention, in the virtual environment. A second advantage is that the central server

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 $\mathcal{Q}$

does not execute any actions, and therefore is free of the game logic. The server merely timestamps actions, queues them for delivery for clients, and manages the network traffic. This allows the server to handle a very large number of clients. Popular systems such as SIMNET [28], [29] and WAVES [30] use similar protocols at the object level — they broadcast updated data objects to all clients.

However, a major drawback of our first action based 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 both at the clients and the server. Thus this first protocol, while ensuring a response time of one round trip and achieving consistency between the clients, has very limited scalability. To achieve better scalability we exploit application semantics, as described next.

B. Using Application Semantics

In the realm of object based protocols, numerous optimizations have been proposed to reduce the number of messages that are sent to each client [6], [7]. Most of these optimizations are variants of *area-of-interest paradigm* described in prior work [31], [32]. In such models, the server restricts the set of update messages (and object data) sent to a client by some syntactic constraint, for example, the visibility of an avatar in the virtual world, a well-known approach. Anyone would therefore naturally consider generalizations of the scalability solutions proposed in these systems to action-based protocols. However, we argue that this approach has a couple of issues 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 what is used to partition the database replicas [6]. 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. Furthermore, in cases like our example of a scrying spell from Section I, there may be no obstruction information at all.

However, using of syntactic constraints such as restricted visibility has a deeper, subtle problem: They are not sufficient for maintaining consistency. For example, none of the current proposals cover transitivity of actions—characters can easily

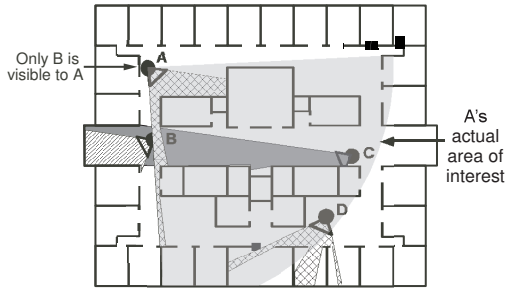

Fig. 2. The RING system limits itself to the visibility of avatars, resulting in an inconsistent state across clients. The actually area that can causally influence A is much larger than its visibility.

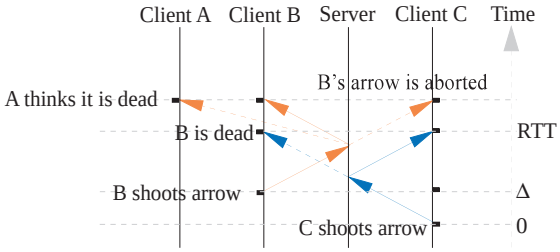

Fig. 3. Inconsistency in area of interest paradigm

interact with one another even if they cannot see one another. We illustrate this problem in Figure 2. Although entities A, B, C, and D (filled circles) all inhabit the same virtual environment, very little interaction (filled and hatched polygons) is possible due to the occlusion of walls (solid lines). In fact, in this example, only two direct interactions are possible — between entity A and entity B; and between entity B and entity C. The restricted vision paradigm suggests that each action submitted by entity B should be sent to clients with entity A and entity C, whereas an action submitted by entity C should be sent only to the client with entity B (because entity C is not visible to entity A). However, this observation leads to an inconsistent state in the system.

Consider the case when entity C shoots an arrow at entity B at time $t = 0$, and entity B shoots at entity A at time $t = \Delta$ where $0 < \Delta < RTT$ (Figure 3). Assuming that the time for the arrow to travel is δ where $0 < \delta < \Delta$, entity B should die before it actually shot the arrow. However, the client with entity B receives entity C's shoot request only at time $t = RTT$, by when it has already sent entity B's shoot request to the workstation with entity A. This client with entity A receives entity B's shoot request at time $t = \Delta + RTT$, and subsequently announces entity A to be dead. It is interesting to note that the client with entity A could have determined entity B's death only if it also knew that entity C had shot entity B.

We therefore conclude that although there is a bound on the visibility of an avatar, the actual area that can influence an avatar is much larger than the visible region (Figure 2). The main reason is that all prior work assumed a *syntactic* restriction on causal influence, however causal influence is really determined by the semantics of the actions in the virtual world. In the next section, we propose exactly such a model

Algorithm 4: Incomplete World 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 executes an action a, it is 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 three possible cases:

4 (Action b originated at some other client, or is a blind write created by the server): 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. In either case, 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 $a_1 \ldots a_i$. 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) $a_{j+1} \ldots a_n$.

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.

Algorithm 6: Transitive Closure(A)

Require: $a_i, \ldots, a_n$ is the action queue
Require: a_{n+1} has just arrived from client C
Require: $+$ denotes prepending an action to a sequence

```
A ← {a_{n+1}}
S ← RS(a_{n+1})
for (j = n; j > i; j = j − 1) do
  if WS(a_j) ∩ S ≠ ∅ then
    if C ∈ sent(a_j) then
      S ← S \ WS(a_j)
    else
      S ← S ∪ RS(a_j)
      A ← a_j + A
      sent(a_j) ← sent(a_j) ∪ {C}
A ← W(S, ζ_S(S)) + A
return A
```

and we show that it can be used to achieve scalability without giving up consistency.

C. The Incomplete World Model

In this section we introduce a novel *semantic-based* action protocol that resolves the consistency problems that we discovered in the previous section. What we learned from the previous section is for a client to determine the effect of an action a, the client needs to have enough information about the virtual world to determine all actions that could potentially influence a. This causal dependence of actions depends on their semantics and frequently cannot be captured by syntactic constraints.

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 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. Armed with these definitions, we can now change our protocols as shown in Algorithms 4, 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 the Incomplete World 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 change, 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. In such cases, it is acceptable to assume that the action was never submitted. The client can

also be optimized for memory. The server can inform the client periodically of the last installed action, enabling the client to garbage collect the results of actions received in the past that it is no longer explicitly interested in. The correctness of our algorithm is stated as follows:

Theorem 1: If the server follows Algorithm 5 and all clients follow Algorithm 4, then in a distributed snapshot of the system the states ζ_{CS} at the clients and the state ζ_S at the server will never be inconsistent.

D. The First Bound Model

Under the Incomplete World Model, each client evaluates only a "necessary" subset of the actions—those actions that actually affect the client. Let us now investigate whether the Incomplete World Model provides any bound on the number of actions each client evaluates.

Suppose a client could evaluate an action set AS in constant time γ independent of the size of AS. Then the time for the server to receive a response for any action from a client would at most be $RTT+\gamma$, where RTT is the round-trip time between the client and the server. This effectively means that the server might need to send to the client all actions that it has seen in the previous $(RTT+\gamma)/\tau$ ticks, which provides our first bound. This discussion assumes that RTT is constant for all clients—we can easily drop this assumption by substituting RTT_{max} for RTT. Assuming that all clients have reasonable latency, and the virtual environment is very large, we believe that this is still a reasonable bound.

The bound is not valid in practice, however, because the number of ticks required for the client program to execute an action sequence AS is, in the worst case, proportional to the number of ticks that AS represents. This worst case comes about when there is at least one action for every distinct tick in AS. Effectively, the time after which the server receives a response for an action becomes $2 \times RTT$, which increases the size of the subsequent AS. The final result of this iterative process is a geometrically increasing size of AS, thereby invalidating the previously obtained bound.

Intuitively, the problem is that when a client submits a new action after having been idle for a while, the server may respond with an unboundedly large AS set, resulting in unacceptably high response time for that client. The solution to this problem is provided by our MMO semantics. Most existing net-VEs have strict properties of locality that we can exploit. Every participant in can be represented as a high-dimensional tuple. Furthermore, this tuple has a finite maximum rate of change in position. Certainly traditional spatial attributes like x, y cannot change more than the maximum object velocity. Similar restrictions apply to attributes like health if the virtual world has a maximum damage amount. As a result, many of the actions are restricted to a ball of fixed radius about a high dimensional point determined by the participant. For example, when a combatant is looking a target to attack, this is ball about the combatant's attack power and spatial position. Therefore, given the position of these balls at time t, and the maximum rate of change, together with an action A of some other participant, we can use simple geometrical calculations to determine if any of the participant's future actions might be directly affected by the outcome of A.

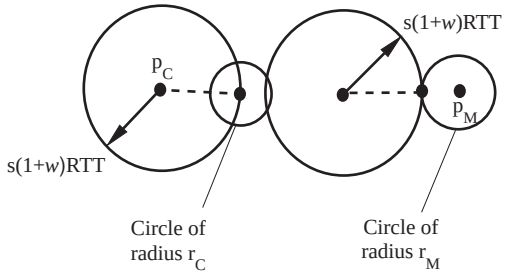

Fig. 4. The worst-case in First Bound Model

The server now works as follows. Instead of waiting for a client to submit an action A and then replying with a set of previous actions that are known to affect A, the server proactively pushes to each client a set AS of all actions that *might* affect that client's future actions, enabling the client to execute the actions of AS during what would otherwise be idle time. More precisely, at regular intervals of $\omega\ RTT$ time, where $0 < \omega < 1$, the server sends to each client C all actions submitted in the previous $\omega\ RTT$ that could possibly affect any of C's future actions.

Claim: The server receives a response for any action A from the client in time $(1+\omega)\ RTT$ of sending A to the client.

Proof: We assume that it takes ½RTT time for an action to travel from the server to the client. Therefore, if an action A (along with some other actions) is sent to the client j ticks after the closest ω RTT cycle from the server, where $j \leq \omega\ RTT/\tau$, it reaches the client j ticks after the client has finished executing the previous action set. The client can therefore execute A in at most j ticks and respond back to the server. The response takes an additional ½RTT. Since j is bounded by $\omega\ RTT$, the maximum time for this entire process is $(1+\omega)\ RTT$.

As stated earlier, the decision whether an action A is sent from the server to a client is based on whether or not the client's future actions could possibly conflict with A. Let the maximum area of influence of A in the virtual world be given by a sphere centered at the point $\bar{p}_A$ and radius r_A. Let the position of the character representing client C be given by $\bar{p}_C$, and let the maximum radius of influence of an action by C be r_C, and let the maximum rate of change in position of any object be given by s. Then A can affect any of C's future action in time $(1+\omega)\ RTT$ if and only if

$$\| \bar{p}_A - \bar{p}_C \| \leq (2s \times (1+\omega)\ RTT) + r_C + r_A \qquad (1)$$

This equation reflects a worst-case in which A affects an object at distance r_A from itself, that object and C's character move towards one another, each traveling at maximum speed s, and they approach to distance r_C within the specified time bound of $(1+\omega)\ RTT$, as illustrated in Figure 4. The equation gives us the first bound on the number of actions that can directly conflict with the actions of the client, represented as a sphere centered at the position of the client in the virtual

world.

E. The Information Bound Model

Though the First Bound Model gives a bound on the number of actions that can directly conflict with a client's actions and therefore have to be sent to the client, the actual set of actions that are sent to a client is the transitive closure of actions that conflict with the aforementioned set of actions.

We claim that the number of uncommitted actions than can directly or indirectly cause a conflict with any given action is unbounded. We illustrate this using the following example.

Dining Philosophers Problem. Consider a scenario with n participants, with each of them trying to grab two forks—one to their left and one to their right. Let them be organized in the form of a circular ring located on earth's equator. If each of them tries to pick up the two forks at the same tick, then although the direct conflicts never involve more than two participants, a transitive closure of conflicts encompasses the entire world.

In order to counter this problem, we believe that the prevalent uncertainty in the system can be used to break the long chains. This can primary be employed to restrict the size of the transitive closure of actions by *dropping* some actions, i.e. declaring some actions as invalid and aborting them immediately upon a submission at the server. An alternate to dropping actions is delaying actions by some amount of time so that the bulk of the actions in the conflicting action set are committed.

The optimal way to drop actions is unclear. For example, fairness becomes an issue when we consider action dropping—what if the actions for a client are repeatedly dropped or delayed? Another issue is to find the optimal set of actions to drop in real-time, especially given the fact that most net-VEs are online and demand immediate response. With more and more people joining net-VEs, a fear in such a protocol is that the cost of evaluating transitive closures of conflicting actions might surpass the cost of processing actions at the server. Evaluating all such techniques is beyond the scope of this paper, and is interesting area for further research.

As a first step towards solving this problem, we propose the Information Bound Model. This model greedily decides whether or not an action should be dropped. Since all clients do not submit actions exactly at the same time, we believe that the random order of arrival of actions at the server will ensure fairness, i.e. the probability of an action getting dropped is the same for all clients. The greedy nature of the algorithm is computationally inexpensive, and therefore we conjecture that the model can be used in real-time environments.

Algorithm 2 gives two important modules of the Information Bound model. The function `onActionSubmission()` is called when any client submits an action. This action is added to a global queue of actions (line 7). The function then evaluates the set of clients (given by *clientConflicts*) that could be interested in the action sometime in the near future (lines 9-14). The second function `onNextTick()` is invoked at every tick, i.e. at regular intervals of time τ. The identifier

Algorithm 7: Information Bound Model

```
1  global actionCount, previousCount, lastCommitted,
   numClients

2  function onActionSubmission(action)
3  begin
4      A_actionCount ← action
5      let i = actionCount
6      for (j = 0; j < clientCount; j+ = 1) do
7          if |p_Ai − p_Cj| ≤ (2s × (1 + ω) RTT) + r_C + r_A
           then
8              clientConflicts_{i,clientConflictCount_i} ← j
9              clientConflictCount_i+ = 1
10         end
11     end
12     actionCount+ = 1
13 end

14 function onNextTick()
15 begin
16     for (i = previousCount; i < actionCount; i+ = 1)
       do
17         let S = RS(A_i)
18         let invalid = false
19         for (j = i − 1; j > lastCommitted; j− = 1) do
20             if isValid_j and S ∩ WS(A_j) ≠ ∅ then
21                 if |p_Ai − p_Aj| > threshold then
22                     invalid ← true
23                     break
24                 end
25                 S ← (S − WS(A_j)) ∪ RS(A_j)
26                 conflicts_{i,conflictCount_i} ← j
27                 conflictCount_i+ = 1
28             end
29         end
30         isValid_i ← not invalid
31     end
32     previousCount ← actionCount
33 end
```

range [*previousCount*, *actionCount*) gives the identifiers of all actions submitted in the previous tick. For each submitted action A, `onNextTick()` evaluates into *conflicts* a transitive closure of all conflicting uncommitted actions. If any of the conflicting actions is at a distance greater than some *threshold* distance from A, then A is dropped.

The First Bound Model and the Information Bound Model together give two bounds. The first bound is on the maximum number of actions that need to be sent to a client due to direct conflicts, represented as a function of time and distance in the attribute hyperspace. The second bound is on the maximum number of actions that can be a part of any actions transitive closure, represented as a function of distance. Combining these two bounds, we get the following (loose) bound on the number of actions sent to a client at each tick, represented as a function

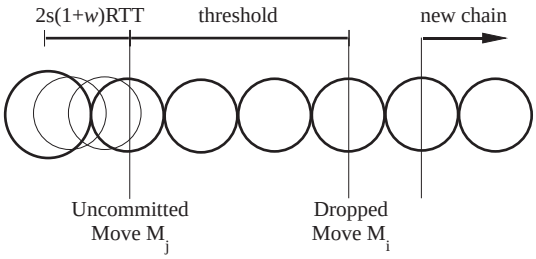

Fig. 5. Chain breaking in the Information Bound Model

of time and distance:

$$\| \bar{p}_A - \bar{p}_C \| \leq (2s \times (1+\omega)\ RTT) + r_C + r_A + \textit{threshold} \quad (2)$$

An important aspect of the Information Bound Model is the conflict detection algorithm. Although virtual worlds require an unordered evaluation of actions with the same timestamp, the decision to drop actions is sequential (lines 19-34). This enables the model to accept a majority of the actions, while dropping only those actions that invalidate the bound. To put things in perspective, we again consider the Dining Philosophers problem. If all participants try to pick up the two forks at the same tick, we conjecture that the decision to drop all of the requests is suboptimal. The primary reason for this is the fact that the intention was to break long chains, and not make a decision. By dropping a few actions at regular intervals, the chain can be broken into numerous pieces, each of which satisfies the requisite threshold.

IV. Optimizing the Protocol

We next give some basic optimizations for our model. Though most of these techniques have been well-researched in graphics and rendering space [33], they also generalize to the domain of event propagation. In particular, as the virtual world is representable by a high-dimensional database, all we have to do is apply many of these techniques to higher dimensions.

A. Inconsequential Action Elimination

Throughout the discussion in this paper, we have assumed that an action submitted by any participant can affect the future actions of all other participants that satisfy a certain bound on the distance between their positions. We claim that the number of such conflicts can be sharply reduced by integrating non-trivial MMO semantics into the system. For example, suppose that a net-VE contains humans and insects. A participant who is pretending to be an insect in the VE would probably need to consistently know the location of other insects and of the humans. However, a participant who is acting as a human in the VE may not need to reliably know the locations of all of the insects. We can therefore extend the system so as to allow the clients to specify exactly what kind of actions and information they are interested in, instead of assuming absolute uniformity.

B. Area Culling

Another assumption that has been made is that the area of influence of any action is a sphere centered at its point of occurrence. However, most of the actions such as shooting an arrow, or even walking, normally have a velocity vector associated with them. Even health may have an associated "velocity" vector to it, if the damage is occurring over time (such the "corrupted blood" disease in *World of Warcraft*). We can therefore integrate this velocity vector in the bound calculation to predict any future conflicts. The conflict equation (Equation 1) can be restructured as:

$$\| \bar{p}_M + (\bar{v}_M \times (t_M - t_C)) - \bar{p}_C \| \leq (2s \times (1+\omega) RTT) + r_C,$$

where $\bar{v}_M$ is the velocity vector associated with M, t_M is the time of occurrence of M, and t_C is the time at which the position of client C was last updated to $\bar{p}_C$. Note that the term, r_M, corresponding to the area of influence of M is now represented as a vector and moved to the left hand side of the equation.

V. Experiments

We built a system implementing the action based protocol in Java 5.0 and conducted experimental studies to quantify and evaluate its performance. We call our implementation *SEVE*, for Scalable Engine for Virtual Environments. We also built an optimized version of a centralized system that represents current online virtual worlds such as Second Life or World of Warcraft. Furthermore, we implemented the NPSNET and the RING architectures, which represent the state of the art in distributed simulations.

Our experimental evaluation is based on a synthetic workload that stresses the consistency issues in MMOs. The synthetic workload is generated by a simple virtual world, similar to the example in Section III-B. We call this virtual world *Manhattan People*. It consists of avatars moving about in a rectangular area and colliding with walls or other avatars. Whenever an avatar bumps into something, it changes its direction by $90°$. By adjusting the number of walls, we can control the computational complexity per action, while the number of participants controls the expected number of conflicts between actions.

A. Experimental Setup

1) System Setup: All performance results were obtained by running the virtual world on an EMULab [34] testbed consisting of 65 machines—64 clients and 1 server. Each EMULab machine was a Pentium III Processor with 2 GB of RAM, running Linux 2.4.0. Timings were obtained using the Java `System.currentTimeMillis()` method. Each machine, except one designated as the central server, was running other programs such as a desktop manager, a document editor and a web browser in the background. We consider this a simple way to emulate a typical client machine. Additionally, we used EMULab to introduce latency at the network level in order to simulate deployment on a wide-area network. The average latency between machines was 238ms. The numbers we present are repeatable, and were averaged over 10 runs of the system, with each run lasting approximately 1 hour.

Virtual world size	1000 x 1000
Number of walls	0 – 100,000
Number of clients	0 – 64
Average latency	238ms
Maximum bandwidth	100Kbps
Moves per client	100
Move generation rate	Every 300ms per client
Move effect range	10units
Avatar visibility	30units
Threshold	1.5 × Avatar visibility

TABLE I
SIMULATION SETTINGS

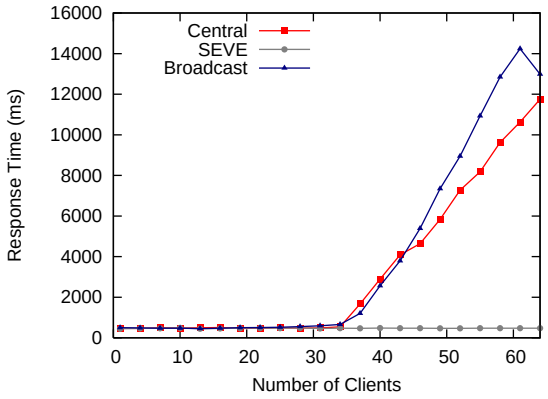

Fig. 6. Scalability of SEVE vs. Central architecture

2) Virtual World Setup: The size of the virtual environment in Manhattan People was fixed at 1000 x 1000 points. Each wall had length 10, and the number of walls was limited to 100,000. Each move evaluation checked for conflicts with a varying number of walls closest to the client's avatar, and all other avatars within walk-able range. Checking for collisions with walls, we made heavy use of trigonometric functions—a complexity that was forced in to simulate the performance of virtual worlds such as Second Life.

As our simulations have shown, the average time required to calculate a single move is as expected linearly related to the number of walls in the virtual world. We omit details on this due to space constraints except that clients required an average of 6.95ms per move, per 1,000 visible walls (1,000 is very close to the average number of walls a client sees for 100,000 walls in our virtual world). Table I gives an overview of the simulation parameters. For our experiments, we varied the number of walls and clients to measure scaling effects.

B. Performance Evaluation

We performed three batteries of 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 NPSNET and SIMNET, the state of the art in distributed simulations; and (c) our action based distributed model (SEVE). Second, we explored the bandwidth requirements of the three models. Third and last, we evaluated the consistency-performance tradeoff.

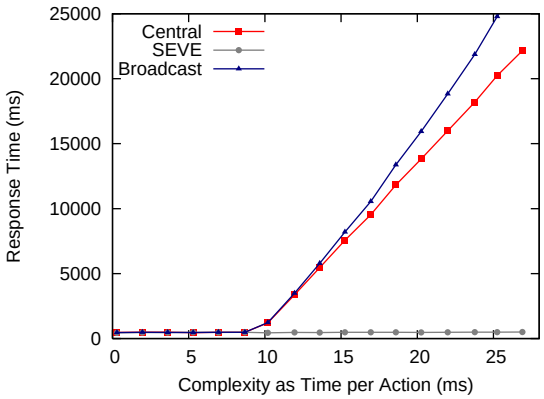

Fig. 7. Response Time vs. Action Complexity

1) Scalability vs. Complexity: For this first set of experiments, every single client submitted a total of 100 moves at intervals of 300ms per move. The number of walls was fixed at 100,000, while we varied the number of clients between 0 and 64. In a single run of the simulation, the number of other avatars that a client's avatar could see was empirically determined to be 6.87 on average.

We empirically determined that the time it took for a machine to evaluate a single move was 7.44ms. Figure 6 compares the response time observed by clients against the number of clients. As apparent from the figure, the centralized architecture and the broadcast model break down at about 30-32 clients. This is not too surprising for the centralized architecture since for every action that a client submits, the server has about 300ms to evaluate it. If 32 clients submit actions simultaneously, each action consuming 7.44ms of a server's time, the total time required to evaluate a round of actions is 240ms. The remaining 60ms can be attributed to synchronization and networking overhead. As noted earlier, each client in the broadcast model has computational requirements comparable to the central server; and therefore we observe a similar scalability for the broadcast model.

In contrast to that, SEVE's response time remained perfectly stable as the number of clients increased. We empirically determined the time for calculating the transitive closure of conflicts over a single move to be 0.04ms on average. However, as the number of clients goes up, so does the number of concurrent moves and the time required to evaluate a transitive closure. This factor is alleviated by the fact that the size of the transitive closure is bounded as a result of the moves getting dropped. We performed experiments on a single server and determined the limit of our implementation to be about 3500 clients.

Figure 7 compares the response time observed by the clients against the time it took to evaluate a single move. The number of clients employed in this experiment was fixed at 25. The centralized model and broadcast model performed well for moves that took less than 10ms for processing. However, as the complexity increased, the response time increased drastically, effectively making the game unplayable. Again, the response time for SEVE remained unaffected.

Finally, we evaluated the sensibility of SEVE with respect

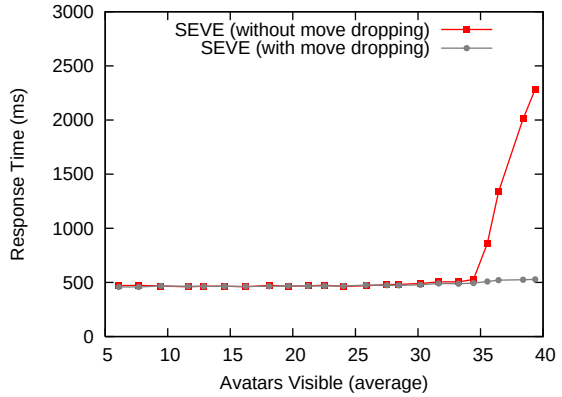

Fig. 8. Effect of increasing density of avatars

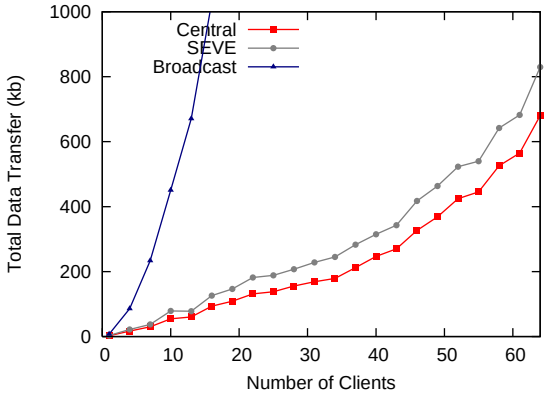

Fig. 9. Total data transfer

Move effect range	1	3	5	7	9	11
% Moves dropped	0	0	0.01	1.53	4.03	8.87

TABLE II

PERCENTAGE OF MOVES DROPPED (VISIBILITY = 20UNITS)

to the density of avatars. Recall that humans are social beings, so avatars can be expected to form clusters in a real system. For this test, the number of clients was fixed at 60. The size of the virtual world was reduced to 250x250 units, and the avatars were initially positioned 4 units apart from each other. We varied the visibility of avatars from 10units to 100units. Figure 8 gives the observed response time versus the average number of other avatars visible to each avatar.

The naive implementation of SEVE bogged down as the number of visible avatars exceeded 35, primarily because the clients ran out of computational power. In comparison, the improved implementation of SEVE started dropping moves that were causing long chains, allowing it to keep response time stable regardless of the density of avatars. The number of dropped moves varied from 1.5%-7.5% for different runs of the system.

At this point, it should be noted that the percentage of moves dropped is in fact independent of avatar visibility. This is because the length of chains depends on the *range of move effect*, and not avatar visibility. Table II gives the percentage of moves dropped as a function of move effect range. While the numbers appear to be fairly high for a large move effect range, the density of avatars in this particular experiment is really an extreme case. We can safely consider this a worst case scenario.

Varying the number of moves per client, or the rate of move generation had no impact on the performance of SEVE. The centralized model and the broadcast model, however, diverged when the number of moves, or the rate of generation, was increased. We omit the corresponding graphs due to space constraints.

2) Bandwidth Requirements: A main concern of distributed systems is in the amount of network traffic generated. Figure 9 shows the comparison between Central, Broadcast and SEVE. As expected, the broadcast model requires excessive network traffic (quadratic in the number of clients). This was the

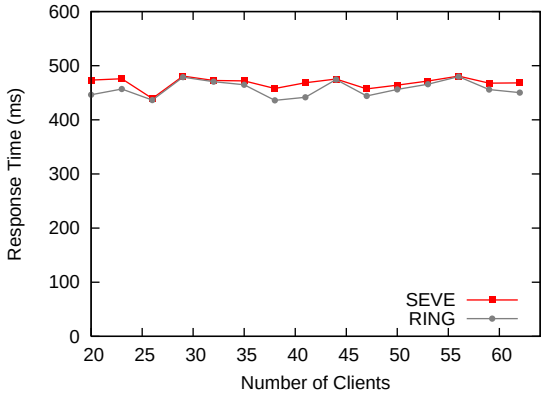

Fig. 10. SEVE vs RING-like Architecture

original reason why systems such as RING were proposed. We note that the total traffic for the server in SEVE does not differ significantly from a centralized model, which obviously is optimal in total traffic. We conclude that SEVE does not incur higher costs on network infrastructure than current systems.

3) Performance vs. Consistency: We evaluated the performance impact of calculating transitive closures in SEVE with 64 clients and 100,000 walls compared to a RING-like architecture which only evaluates actions within the visible range of an avatar. The average number of avatars that an avatar could see was increased to 14.01 as opposed to 6.87 earlier, leading to more conflicts processing at the clients. Figure 10 shows the results we obtained.

Calculating the transitive closure in SEVE accounted for a runtime overhead of 1% compared to the RING-like architecture. This shows that the runtime overhead of our strongly consistent approach is negligible.

In summary, our experiments show that our architecture is massively scalable while preserving strong consistency. It gives an order of magnitude improvement over existing strongly consistent architectures for networked virtual environments.

VI. RELATED WORK

Recent work in commercial MMOs has introduced the idea of dynamic zoning [4]. While dynamic zones are more flexible than traditional zones, they still restrict player actions

to a geographic area. Reality Build For Two [5] and MR Toolkit [8] 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 single state change. This approach yields $O(N^2)$ update messages during every simulation step, and this does not scale. NPSNET [35] 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 from each client is the same in MR Toolkit. RING [6] and DIVE [7] 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 would not be interested in a particular update. This approach takes these frameworks very close to our model. However, 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. [36], [37] evaluate replication techniques based on broadcasting events in total order. Our algorithm can be understood as a fast-paced instance of 2-tier replication [38], which in turn builds on multi-version serializability theory [39]. Content-based publish-subscribe [40] is a generalization of our perceptions-as-continuous-queries model, however the focus is on data distribution and not consistency [41].

VII. Conclusions

In this paper we motivate that at the core of networked virtual environments lie data management problems. We identified an interesting concurrency problem to which we proposed a novel practical solution based on taking semantics into account. We believe, however, that we just scratched the surface of this (for the database community) new area, and that both virtual worlds as well as other virtual networked environments — from collaborative problem solving to online games — can benefit from solutions from the database community for years to come.

References

[1] Microsoft Corp, "http://www.microsoft.com/esp."

[2] S. Carless, "The Activision/Blizzard merger." [Online]. Available: http://www.gamasutra.com/php-bin/news_index.php?story=16458

[3] T. Keating, "Dupes, speed hacks and black holes: How players cheat in MMOs," in *Proc. Austin GDC*, 2007.

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

[5] 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*.

[6] T. A. Funkhouser, "RING: A client-server system for multi-user virtual environments," in *Proc. SI3D*, pp. 85–92, 209, 1995.

[7] 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*.

[8] C. Shaw and M. Green, "The MR toolkit peers package and experiment," in *VR*, pp. 463–469, 1993.

[9] W. White, A. Demers, C. Koch, J. Gehrke, and R. Rajagopalan, "Scaling games to epic proportions," in *Proc. SIGMOD*, pp. 31–42, 2007.

[10] R. Bartle, *Designing Virtual Worlds*. New Riders Games, 2003.

[11] 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.

[12] Wikipedia, "Instance dungeon." [Online]. Available: http://en.wikipedia.org/wiki/Instance_dungeons

[13] R. R. Koster, "From instancing to worldy games." [Online]. Available: Raph Koster's Blog: http://www.raphkoster.com

[14] Tobold, "Servers and critical mass." [Online]. Available: Tobold's MMORPG Blog: http://tobolds.blogspot.com

[15] A. Taylor, "The problem with world of warcraft." [Online]. Available: Wonderland Blog: http://www.wonderlandblog.com

[16] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.

[17] K. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems," in *Proc. SOSP*, pp. 123–138, 1987.

[18] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Trans. Comput. Syst.*, vol. 17, no. 2, pp. 41–88, 1999.

[19] P. Kabus, W. W. Terpstra, M. Cilia, and A. P. Buchmann, "Addressing cheating in distributed mmogs," in *Proc. NetGames*, pages 1–6, 2005.

[20] A.-R. Sadeghi and C. Stüble, "Taming "trusted platforms" by operating system design," in *WISA*, 2003, pp. 286–302.

[21] A.-R. Sadeghi and C. Stüble, "Property-based attestation for computing platforms: caring about properties, not mechanisms," in *Proc. NSPW*, pp. 67–77, 2004.

[22] Sun Microsystems, "Project Darkstar." [Online]. Available: http://www.projectdarkstar.com

[23] 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.

[24] B. Ng, A. Si, R. W. Lau, and F. W. Li, "A multi-server architecture for distributed virtual walkthrough," in *Proc. VRST*, pp. 163–170, 2002.

[25] R. Kazman, "Making WAVES: On the design of architectures for low-end distributed virtual environments," in *VR*, pp. 443–449, 1993.

[26] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: a distributed architecture for online multiplayer games," in *Proc. NSDI*, pp. 12–12, 2006.

[27] 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.

[28] H. A. Taha, "Introduction to SIMNET v2.0," in *Proc. WSC*, 1988.

[29] J. M. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen, "The SIMNET virtual world architecture," in *VR*, pp. 450–455, 1993.

[30] R. Kazman, "Making WAVES: On the design of architectures for low-end distributed virtual environments," in *VR*, pp. 443–449, 1993.

[31] 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.

[32] G. Morgan, F. Lu, and K. Storey, "Interest management middleware for networked games," in *Proc. I3D*, pp. 57–64, 2005.

[33] I. Pantazopoulos and S. Tzafestas, "Occlusion culling algorithms: A comprehensive survey," *J. Intell. Robotics Syst.*, vol. 35, no. 2, 2002.

[34] 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.

[35] 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.

[36] M. Wiesmann, "Comparison of database replication techniques based on total order broadcast," in *IEEE TKDE*, pp. 551–566, 2005.

[37] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems," in *Proc. ICDCS*, p. 464, 2000.

[38] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proc. SIGMOD*, pp. 173–182, 1996.

[39] P. A. Bernstein and N. Goodman, "Concurrency control algorithms for multiversion database systems," in *Proc. PODC*, pp. 209–215, 1982.

[40] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[41] E. S. Liu, M. K. Yip, and G. Yu, "Lucid platform: applying hla ddm to multiplayer online game middleware," *Comput. Entertain.*, vol. 4, no. 4, p. 9, 2006.