Survey of State Melding in Virtual Worlds

HUAIYU LIU and MIC BOWMAN, Intel Labs, Intel Corporation
FRANCIS CHANG, Portland State University

The fundamental goal of virtual worlds is to provide users with the illusion that they are all seeing and interacting with each other in a consistent world. State melding is the core of creating this illusion of a shared reality. It includes two major parts: consistency maintenance and state update dissemination. Well-designed state melding technologies are also critical for developing a virtual world that can scale to a large number of concurrent users and provide satisfying user experiences. In this article, we present a taxonomy of consistency models and categorization of state update dissemination technologies for virtual worlds. To connect theories and practices, we then apply the taxonomy to case study several state-of-the-art virtual worlds. We also discuss challenges and promising solutions of state melding in large-scale virtual worlds. This survey aims to provide a thorough understanding of existing approaches and their strength and limitations and to assist in developing solutions to improve scalability and performance of virtual worlds.

Categories and Subject Descriptors: A.1 [**General Literature**]: Introductory and Survey; c. [**Computer Systems Organization**]

General Terms: Design, Performance

Additional Key Words and Phrases: State melding, virtual worlds, consistency maintenance, consistency models, state update dissemination, scalability

ACM Reference Format:
Liu, H., Bowman, M., and Chang, F. 2012. Survey of state melding in virtual worlds. ACM Comput. Surv. 44, 4, Article 21 (August 2012), 25 pages.
DOI = 10.1145/2333112.2333116 http://doi.acm.org/10.1145/2333112.2333116

1. INTRODUCTION

Virtual worlds are computer-based simulated 3D environments for users to explore, collaborate, and interact in real time. There are two broad classes of popular virtual worlds: general-purpose virtual worlds, such as Second Life®,[1] and massively multiplayer online. games, such as World of Warcraft™[2] and Eve Online®.[3] Virtual worlds have been growing rapidly in the past a few years. It is estimated that there are 137 million virtual world users today, with up to 1 billion projected users in the year 2017 [Virtual Worlds News 2008].

Typically, a virtual world has the following features [Singhal and Zyda 1999]: a shared sense of space, a shared sense of presence, a shared sense of time (real-time interaction), a way to communicate (by gesture, text, voice, etc.), and a way to share information and manipulate objects. The fundamental goal of virtual worlds is to provide

[1] http://secondlife.com/.
[2] http://www.worldofworcraft.com/.
[3] http://www.eveonline.com/.

Author's email address: H. Liu (corresponding author), huaiyu.liu@intel.com.

DOI 10.1145/2333112.2333116 http://doi.acm.org/10.1145/2333112.2333116

users with the illusion that they are all seeing and interacting with each other in a consistent virtual world in a fashion similar to how people in a space in the real world experience a consistent reality. The core of creating this illusion of a shared reality is to meld together the actions of all the objects, avatars, and their interactions into a form which can be shared among all of the participants [Second Life Wiki 2008]. State melding is about creating and sharing that melded state.

In general, there are two major parts in state melding: (1) consistency maintenance, which deals with how to create and update the dynamic shared state on participating processes, and (2) dissemination of state updates, which deals with how to utilize resources efficiently to disseminate the state updates to the participating processes. Well-designed state melding technologies not only create the illusion of a shared reality but also are critical for developing a virtual world that can scale to a larger number of concurrent users and provide satisfying user experiences with complex scenes and high-fidelity interactions. In this article, we review different state melding technologies that have been developed in the past, discuss how each technology may affect the scalability and performance of virtual worlds, and apply our discussions to several state-of-the-art virtual worlds. We believe that a thorough understanding of existing approaches and their strengths and limitations will assist developing solutions to improve scalability and performance of virtual worlds.

In short, the contribution of this article is three-fold.

—We present a taxonomy of consistency models and a categorization of state update dissemination technologies that are applicable to virtual worlds and analyze the strength and limitations of different models and technologies.
—To connect theories and practices, we apply the taxonomy and categorization introduced in this article to several state-of-the-art virtual worlds, including well-known systems, such as Second Life, Eve Online, and Darkstar, to analyze the consistency model that is supported by each system as well as the update dissemination technologies used.
—We discuss the challenges of state melding in large-scale virtual worlds and provide insights into promising directions.

The remainder of the article is organized as follows. In Section 2, we review several consistency models that are applicable to virtual worlds, compare the strength and limitations of each model, and identify protocols to support each model. In Section 3, we focus our discussion on different approaches to efficiently disseminating state updates. In Section 4, we take a closer look at a few state-of-the-art virtual world applications and analyze the protocols they use for state melding. In Section 5, we discuss the challenges of state melding in large-scale virtual worlds and promising approaches in addressing the challenges. Finally, we conclude the survey in Section 6.

2. CONSISTENCY MODELS

When building a virtual world, a key consideration is how to ensure that the participating processes keep a consistent view of the shared state of the virtual space. This is fundamental to providing users with the illusion that they are all experiencing the same events and interacting with each other in the same virtual space. It is a great challenge to maintain a consistent view for users who are interacting concurrently, especially to support many concurrent users and update each user's customized view with unnoticeable delay. In the best case, inconsistency may just lead to transient visible artifacts. In other cases, it may cause more serious problems. In fact, consistency violation is a major source of security problems in virtual worlds [Keating 2007].

Consistency models has been introduced and studied in many distributed applications, including distributed shared memories [Lamport 1979; Welch 1994; Raynal

and Schiper 1996], databases [Bernstein et al. 1987], distributed computing [Lamport 1978], and computer-supported cooperative work (CSCW) [Ellis and Gibbs 1989]. In general, a consistency model is concerned with the causal order of operations in different processes. The concept of operation is somewhat different in different applications. In distributed shared memories, an operation refers to a single read or write. In databases, an operation is a transaction that consists of multiple reads or writes. In distributed systems, an operation (or an event) usually refers to the sending and receiving of a message. In our discussion, we follow the concept of the *causal order* of events as defined in Lamport [1978]. Further, for virtual world operations, we define an *event* as sending or receiving of a state update message or applying a state update (i.e., updating the state in the local replication).

Compared to traditional distributed applications, virtual worlds typically exhibit the following characteristics of consistency and interactivity [Bouillot and Gressier-Soudan 2004; Delaney et al. 2006].

—*Causality.* The "happening before" relation introduced by Lamport [1978].
—*Concurrency.* The simultaneous execution of events by different users on the same entity.
—*Simultaneity.* If two events are perceived by a user simultaneously, then they are perceived by other users simultaneously as well. It refers to the strong requirements on the temporal relationship of events.
—*Instantaneity (real-time responsiveness).* For a targeted event and from a human user point of view, the time between the event being generated and the time it is executed and perceived by the user is unperceivable.

Causality and concurrency are well-known properties of distributed applications. It is simultaneity and instantaneity that are the desired properties that imitate the real-world interactions and distinguish virtual worlds from other distributed systems. In fact, due to the strong requirement for instantaneity, virtual world implementations typically would tolerate some short-term inconsistency in order to provide fast responsiveness to user inputs (more discussions in Section 2.3). Hence, when we say a virtual world system supports a consistency model, it means that the system always tries to maintain the consistent state as defined by the model, and it is able to detect short-term consistency and take actions to converge back to the desired consistent state.

In general, there are two main categories of consistency models that can be applied to virtual worlds: ultimate consistency and deadline-based consistency. Each of them has different capabilities of supporting the preceding characteristics. In a deadline-based consistency model, consistency needs to be achieved by time $t + \Delta$ after some event happens at time t, for a predetermined amount of delay Δ. By contrast, in ultimate consistency models, it takes an undetermined amount of time to achieve consistency. A set of formal definitions of some consistency models can be found in Raynal and Schiper [1996]. Bouillot and Gressier-Soudan [2004] have also done a good job in categorizing consistency models.

Consistency in virtual worlds is context specific (depending on the type of interactions), and different types of virtual worlds have different requirements. A virtual world designer needs to understand the interaction context, its requirements on different aspects of consistency, and the strength and weakness of each consistency model so as to apply the appropriate models. In this section, we present a taxonomy of consistency models. We also analyze the strength and weakness of each model, the consistency protocols to support the models, and the application of the consistency models and protocols to virtual worlds.

2.1. Ultimate Consistency Models

In ultimate consistency models, there is little consideration of the particular wall-clock time when the operations would be executed. They only impose certain rules or certain orders for operations to be executed.

—*Causal consistency [Lamport 1978].* This requires that all the causally related events be perceived in the same order by all processes, while unrelated concurrent operations may be perceived in different orders by different processes.
—*Sequential consistency [Lamport 1979].* This model is first discussed in distributed shared memories. It requires that the result of any execution is the same as if the operations were executed in some sequential order that is consistent with the order seen at individual processes. (That is, if event A happens before event B on one process, that order is preserved in the sequential order.)
—*Serializability [Bernstein et al. 1987].* This requires that all transactions appear to have executed atomically in some sequential order that is consistent with the order seen at individual processes. This model has been widely used in databases. Note that if every transaction is reduced to a single read or write operation, serializability is the same as sequential consistency [Raynal et al. 1997].

Although these models do not have built-in mechanisms to achieve low response time, in practice, they are often used in virtual worlds because they have been widely studied. In such virtual worlds, additional mechanisms are applied to reduce response delays, such as dead-reckoning (more discussions in Section 2.3.2 and Section 4).

2.2. Deadline-Based Consistency Models

In deadline-based consistency models, an operation is time stamped using real time instead of logical time. Such a model imposes real-time constraints on when an operation should be executed. By "an operation is executed in a process", we mean that an update message is received and that the update is applied to the replicated state maintained by the process. In our discussion, we assume the clocks of all processes are synchronized with limited clock drifting, for instance, by running the Network Time Protocol.[4]

—*Perceptive consistency [Bouillot and Gressier-Soudan 2004; Qin 2002].* If any operation is executed on process i at time t, it must be executed on process j at t as well. At a high-level point of view, it indicates that for a group of users interacting with each other, if a user takes an action, then the action will be perceived by everyone simultaneously (though the action might be perceived at a time later than the time at which it was taken by the user). It is also referred to as *absolute consistency* [Qin 2002].
—*Delayed consistency [Qin 2002].* This is a relaxation of perceptive consistency. In this model, for any operation issued by host i, say at time t, it takes effect instantaneously on host i. On a remote host j, it is executed at time $t + \Delta(i,j)$, where $\Delta(i,j)$ is the phase different between processes i and j, and it is a parameter specific to these two processes. Unlike perceptive consistency, this model does not force all processes to execute the same operation at the same time. Rather, it makes sure that the state of objects operated by one process be viewed at another process with a consistent delay.
—*Timed consistency [Torres-Rojas et al. 1999].* This guarantees that a local action at time t will be perceived by others at a time bounded by $t + \Delta$. It includes timed sequential consistency (TSC) and timed causal consistency (TCC) by combining the

[4]http://www.ntp.org/.

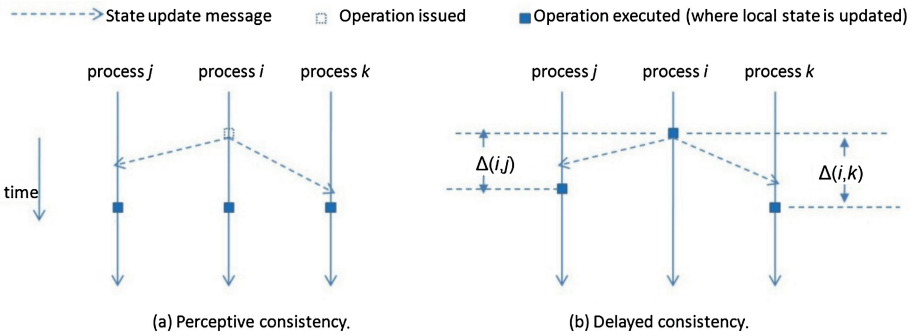


Fig. 1. Illustration of operation executions.

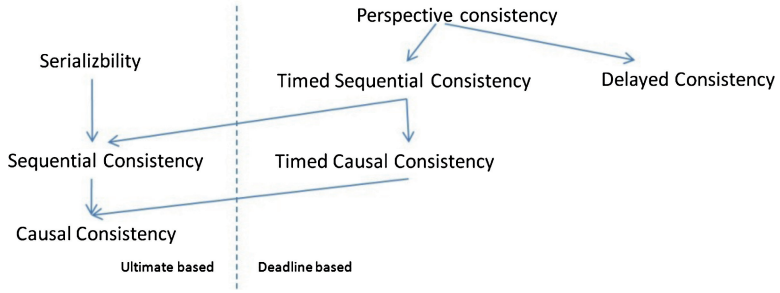


Fig. 2. Relationship of consistency models.

requirements of sequential consistency and causal consistency with the executed-on-time requirement (defined next).

Figure 1 illustrates perceptive consistency and delayed consistency. In Figure 1(a), an operation is issued on process i at time t, and it is executed at time $t+\Delta$ by all processes (where Δ is the same for all). In Figure 1(b), the requirement that Δ is the same for all processes is relaxed, and each process may executed by the same operation at a different time, as long as on each process, say process j, the operation issued by process i is always executed by a delay of $\Delta(i,j)$. The parameter $\Delta(i,j)$ could be specified based on user requirements. Both perceptive consistency and delayed consistency specify the exact execution time of an operation. We refer to it as the *executed-at-specified-time* property. Timed consistency models, namely TSC and TCC, only require the operation be executed on all processes no later than time $t+\Delta$, or the *executed-on-time* property. If Δ is infinite, then TSC (or TCC) is effectively equivalent to sequential consistency (or causal consistency). Also, in each of the deadline-based consistency models previously discussed, if Δ (or $\Delta(i,j)$ in delayed consistency) is less than the perceptive threshold, then the model is said to exhibit properties of *instantaneity*.

Figure 2 summarizes all the consistency models discussed in Sections 2.1 and 2.2. In the figure, an arrow from a model to another indicates that the former has stronger consistency constraints than the latter.

2.3. Protocols Implementing Consistency Models

A number of protocols have been developed to implement different consistency models. In general, they can be categorized based on their approaches in addressing a fundamental trade-off in virtual worlds, the *consistency-responsiveness trade-off*. From a

user's perception, response time refers to the delay between the time a user issues an operation and the time the operation's effect is observed at the user's local machine. If the response time exceeds a perceptive threshold, users will notice the delay. Unfortunately, due to inevitable network delays in delivering updates, optimization of response time and avoidance of short-term inconsistency are conflicting goals [Singhal and Zyda 1999].

In general, there are two types of approaches to address the consistency-responsiveness trade-off and manage the time that an operation is executed on each process (referred to as *time management*): conservative approaches versus optimistic approaches [Fujimoto 2003]. Time management is a well studied topic in parallel discrete event simulation systems (PDES) [Fujimoto 1999]. In PDES, time management typically not only ensures that events are processed in correct orders but also that a simulation with the same inputs produces exactly the same results. Virtual worlds usually do not require repeatable simulations and sometimes may even tolerate incorrect event ordering. Hence, some concepts developed in PDES have been applied to virtual worlds by relaxing the requirements.

2.3.1. Conservative Approaches. These approaches delay execution of local operations until they are supposed to be consistent on all processes. Unfortunately, delaying local operations reduces responsiveness. Hence, added delays should be chosen according to the application's semantics, since user's perceptive thresholds differ for different media and different types of interaction.

—*Local lag [Mauve 2000].* When an operation is issued at time t, its timestamp for the execution will be $t + \Delta$. Both the local process and remote processes will execute the operation at time $t + \Delta$. The delay Δ introduced for the local user is called a *local lag*. For the amount of local lag between two processes, a minimal delay was chosen based on the network delay between the two processes, while the maximal delay (the longest acceptable response time) was chosen based on human perception and the type of application. The local lag can then be be chosen between the minimal and maximal delays to adapt to the specific consistency requirements. A repair mechanism is needed to accommodate operations that arrive late due to network delays. This approach could be applied to achieve perceptive consistency.

—*Bucket synchronization [Gautier and Diot 1998].* In this protocol, time is divided into intervals, or bucket lengths of T (e.g., 25 buckets per second, or $T = 40$ ms). Each update event is assigned to a bucket and time stamped accordingly. The local view of the global state is calculated using all the local updates as well as all the remote updates whose timestamps are within interval $[t - \Delta, t - \Delta + T]$, where $[t, t + T]$ is the current time interval, and Δ is the synchronization delay (100 ms as in [Gautier and Diot 1998]). An update message is discarded if the transmitting delay is more than Δ. Note that this approach is similar to a playout buffer mechanism [Ramjee et al. 1994] used in audio streaming.

2.3.2. Optimistic Approaches. Such an approach is optimistic in the sense that it assumes that few inconsistencies will occur. In predictive time management, future events are predicted and distributed before they actually occur. In other approaches, local actions take effect instantaneously to achieve better responsiveness, and inconsistencies at remote processes are allowed to occur. Convergence is done ultimately by conflict resolution or reconciliation schemes. Optimistic approaches are often applied in models with relaxed consistency constraints, such as late consistency and timed consistency. However, short-term inconsistency can occur due to network delays or incorrect predictions. Hence, correction or rollback mechanisms need to be applied when short-term inconsistency is discovered.

—*Predictive Time Management*. This mechanism was first proposed in PaRADE [Roberts 1996]. Certain future events can be predicted, and both an event and its expected time of occurrence are transmitted before the event actually occurs. For instance, events such as interaction with an entity may be predicted through heuristics of collision prediction or knowledge of intent. Knowledge of network delays is used to predict and send out an event so that it can be delivered to all participants prior to its commencement time. Finally, rollback strategies must be provided to negate incorrectly predicted events. The execution of local events can also be delayed so that the same event will execute at the same time across the system. This approach can be applied to virtual worlds where the state of the relevant objects is small (e.g., position and velocity) and relatively easy to predict, with each object having only one controller. It becomes less appropriate when the state is complex and objects are operated by multiple users.

—*Time Warp*. This is a mechanism designed to implement virtual time, initially proposed by Jefferson [1985] to support distributed discrete event simulation and distributed database. Virtual time is ticked by an imaginary global virtual clock and used to measure computational progress and define synchronization. It always progresses forward. Time warp operates by executing each message as soon as it arrives (optimistically). If a message x arrives at a process and has a timestamp earlier than a message already executed, the process undoes all the events back to the time dictated by the timestamp of x (rollback) and restarts execution from that time. It must also send messages to undo any incorrect outputs that were communicated to other nodes (rollback propagation). A main limiting feature of TimeWarp is the requirement of checkpointing the execution context at every message. In addition, TimeWarp issues a rollback immediately upon detecting a late event and suffers from excessive rollbacks. To overcome the limitations, several schemes have been proposed, including Breathing Time Warp [Steinman 1993] and Trailing State Synchronization (TSS) [Cronin et al. 2004]. In Breathing Time Warp, optimistic executions are limited to events within an event horizon. Events beyond the horizon cannot be guaranteed to be consistent and are therefore not executed. Trailing State Synchronization was designed for distributed first-person shooter games, which are the most latency-sensitive multiplayer games. Instead of keeping state snapshots at every event, as in TimeWarp (or every few events in the Breathing algorithm), TSS keeps multiple copies of the same game state, where each copy will execute all events with a different synchronization delay. Only the leading state—the copy with the shortest synchronization delay—is rendered to the screen, while the other trailing states are used to detect and correct inconsistencies. Each trailing state will see fewer mis-ordered events than the state preceding it by waiting longer for delayed events to arrive before executing. If an inconsistency is discovered, a rollback from the incorrect leading state to the correct trailing state is performed.

—*Dead Reckoning*. This approach has been widely applied in distributed interactive simulations (DIS) [IEEE 1993], where object behavior over time is "known". Examples include an airplane flying with a constant velocity or a projectile following a ballistic trajectory. In this approach, participating processes agree on a prediction algorithm, an error threshold, and a convergence algorithm. An object's current state is computed based on previously received updates. No object updates are sent until the difference between the predicted state and the real state exceeds a predefined threshold. Convergence is handled by an agreement between participants that determine how to correct inconsistencies once new updates are received. Similarly to predictive time management, dead reckoning can be applied for the state that is small (e.g., position and velocity of an object) and relatively easy to predict, and each object has only one controller.

Table I. Comparison of Ultimate Consistency Models

Consistency Model	Support of Interaction Properties	Protocols	Examples of Virtual World Systems
Causal Consistency	Preserves causality. Does not resolve concurrency efficiently, and resolution schemes (e.g., versioning, convergence) need to be added to solve concurrency and user-intention. Nondeterministic responsiveness.	Wait-free implementations [Ahamad et al. 1991; Raynal and Schiper 1995].	Real-time collaborative graphics editors [Sun and Chen 2000].
Sequential Consistency	Supports concurrency. No strong support of simultaneity due to lack of temporal relationship of executions. Nondeterministic responsiveness.	Centralized solution: central server imposes total ordering [Greenhalgh et al. 2000], locking [Newman-Wolfe et al. 1992], serialization [Kanawati 1997]. TimeWarp [Jefferson 1985]	Distributed conferencing system [Newman-Wolfe et al. 1992], MASSIVE-3 [Greenhalgh et al. 2000], Second Life (see discussions in Section 4).
Serializability	Same as above.	Two-phase commit [Moss 1985].	Darkstar [Waldo 2008] (where serializability is achieved across game servers)

Orthogonal to these approaches, a few other mechanisms have also been proposed to address the consistency-responsiveness trade-off. One is to identify and label different types of operations to apply different consistency maintenance protocols within the same virtual world [Greenhalgh et al. 2000]. Another approach is to accept the existence of inconsistency and introduce a metric to evaluate and keep the inconsistency in control [Zhou et al. 2004].

2.4. Comparison of Consistency Models and Protocols

To summarize, Tables I and II compare different consistency models, the virtual world applications that they are able to support, and the protocols that have been designed to support each model.

Ultimate consistency models have been widely studied in different distributed systems, and various protocols have been developed to support these models. In general, they are better understood than deadline-based consistency models. However, due to the lack of constraints on the wall-clock time at which each operation is executed, ultimate consistency models are usually only applied to virtual worlds that do not have strong requirements for simultaneity or instantaneity, such as virtual real-time graphic editors and distributed conferencing. Another approach in applying ultimate consistency models is to categorize events into classes with different requirements on preserving causal relationships and apply different protocols to different classes [Roberts and Sharkey 1997] (e.g., the order of hitting a ball and the ball's movement afterwards need to be preserved, but orders of different avatar's random movements may not).

In the deadline-based consistency models, protocols designed for achieving perceptive consistency introduce local delays in executing operations to prevent misordering. Such protocols support simultaneity but have limitations in supporting latency-sensitive applications. Hence, they are often applied to an application where optimizations could be applied due to the specific characteristics of the application. For example, in the distributed multiplayer game MiMaze [Gautier and Diot 1998], where bucket synchronization is used, movements are limited to a confined maze. In contrast, delayed

Table II. Comparison of Deadline-Based Consistency Models

Consistency Models	Support of Interaction Properties	Protocols	Examples of Virtual World Systems
Perspective Consistency	Provides simultaneity (i.e., allow users to perform tightly coupled or synchronous actions), concurrency management, causality, and deterministic responsiveness.	(1) Local lag [Mauve 2000]. (2) Bucket synchronization [Gautier and Diot 1998]. (3) Predictive Time Management [Roberts 1996].	TeCo3D [Mauve 1999], MiMaze [Gautier and Diot 1998], PaRADE (an Arena application) [Roberts 1996].
Delayed Consistency	Provides concurrency management, causality, and deterministic responsiveness, but not simultaneity.	(1) Conductor-driven synchronization [Bouillot 2004].	Distributed musician interaction [Bouillot 2004].
Timed Consistency (TSC & TCC)	Simultaneity not provided. Suited for concurrency resolution and responsiveness control, but not for continuous media having strong temporal relationships between two successive actions.	(1) Lifetime-based protocol [Torres-Rojas et al. 1999]. (2) Variations of TimeWarp: Breathing Time Warp [Steinman 1993], Trailing State Sync [Cronin et al. 2004]. (3) Dead reckoning with periodic synchronization [Lui et al. 1999].	Multimedia real-time collaborative applications. Eve Online (http://www.eveonline.com/.)

consistency and timed consistency models achieve low response time (local operations are executed immediately) but in general are not appropriate for interactions that need strong support of temporal relationships, such as simultaneity.

3. STATE UPDATE DISSEMINATION

To achieve consistency, when one object inside a virtual world performs an operation, the replicated copies on all participating processes need to be updated to reflect the result of the operation. When the number of concurrent users or the scene complexity increases, the network quickly can be overwhelmed by the aggregated traffic of delivering updates. In particular, the aggregated traffic increases quadratically with the number of concurrent users [Liu et al. 2010]. For example, suppose there are N users, each having an avatar in a virtual world. When one avatar moves, every one of the other $N-1$ users needs to receive an update of the moved avatar's new position (the update could be either sent in a peer-to-peer fashion or channeled through a server). When all the avatars move to new positions, $N(N-1)$ updates need to be transmitted in the network. Figure 3 shows our measurements of the total outgoing traffic with an increasing number of users connected to a virtual world server [Liu et al. 2010]. In this experiment, each client had an avatar in the world, and the avatar moved around following the Random Waypoint module [Johnson and Maltz 1996]. If two avatars collided with each other, they would change walking directions. The result was from running the OpenSimulator 0.6.3, an open-source virtual world server platform that is compatible with Second Life viewers.[5] (The workload can be downloaded from ScienceSim.)[6] The result clearly shows the trend of quadratic increase in the amount of aggregated network traffic.

Figure 3 shows the baseline scenario for the amount of network traffic, where every avatar's movement update is sent to every other user interacting in the space. To

[5]http://opensimulator.org/wiki/Main_Page.

[6]http://www.sciencesim.com/wiki/doku.php.

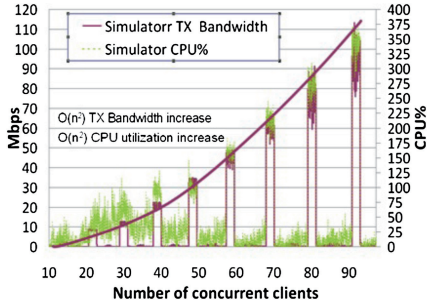


Fig. 3. Bandwidth utilization versus increasing number of concurrent users.

address this communication bottleneck, exploring redundancies in updates to reduce network traffic is critical. One important observation is that each user has a unique perspective of the world, given that each user has his own view point to observe the world. Hence, a user may not need to receive all updates from the space he is interacting in, and for updates the user needs to receive, he may not need to receive all of them in full fidelity. In addition, for users sharing similar perspectives (e.g., users whose viewpoints are close to each other), some updates sent to each of them are quite similar. Different technologies have been developed to explore such communication redundancies. Among them, interest management refers to a group of techniques that try to identify what are necessary updates for each user and only send these updates. Some also explore redundancy in users' shared perspectives in order to apply multicasting to reduce network traffic. Levels-of-detail is another type of technique that explores the redundancy in updates of the same object and trade off fidelity of some updates for reducing network traffic and enabling longer view distances.

3.1. Interest Management

The objective of interest management is to reduce the number of state updates that need to be sent to a given process by sending only the updates relevant to the process (e.g., sending only visible updates to remote viewers). The identification of "only the relevant updates to the process" is subject to the system's policy. Interest management can be abstracted using a publish-subscribe model in which publishers are objects that produce updates, and subscribers are objects that consume updates. This model of update dissemination requires the world to implement mechanisms to allow subscribes to discover publishers and to subscribe and unsubscribe to their updates. Various interest management techniques have been proposed which can be mainly categorized as spatially-based and class-based approaches. Generally, spatially-based interest management is determined based on the relative position of objects in a virtual world, while class-based interest management is based on an object's variables (e.g., type).

3.1.1. Spatial-Based Interest Management.

Aura-based filtering. This technique is based on the focus-nimbus model [Benford and Fahlén 1993], where *focus* and *nimbus* are defined to describe the awareness between two objects. The focus of an object represents a subspace within which the object can perceive, and *nimbus* represents a subspace across which an object can be perceived by others. The aura of an object can be understood as an approximation of the union of the object's focus and nimbus. Conceptually, objects carry their auras with them. When they move through space and when two auras collide, interaction between the objects becomes a possibility. In this model, an object's update only needs to be sent to other objects whose auras intersect with its own. Computation of the

intersection of auras is also called interest filtering. The pure focus-nimbus approach has been implemented in multiple systems including MASSIVE-I [Greenhalgh 1998]. Its advantage is that it allows fine-grained interest management and is especially suitable where there is a connection for each client with the server. However, it does not scale well due to the cost of computing the intersection of auras, which is of $O(mn)$, with m being the number of subscribers and n being the number of the publishers.

Region-based filtering. Region-based filtering is an approximation of the pure focus-nimbus model. In this approach, the world space is partitioned into regions. Regions that intersect the area of interest (focus) of a subscriber form an area of subscription from the union of the intersected regions. The area of subscription represents an approximation of the true area of interest of the subscriber but is easier to compute. The quality of the approximation is highly dependent on the shape and size of the regions. In Spline [Barrus et al. 1996], the world is divided into regions which can be of any shape or size. Yet, it has been shown that hexagons can better approximate the precision of the pure focus-nimbus model than regular square tiles. They have been used in systems such as NPSNET [Macedonia et al. 1995] and in the communication model presented in Fieldler et al. [2002]. More recently, triangular partitioning of virtual space was proposed by Boulanger et al. [2006], which is shown to be successful at culling update messages and greatly reducing the computational requirements of spatial-based filtering comparing to square and hexagonal partitioning.

3.1.2. Class Based Interest Management. In class-based filtering, every class of objects that will participate in a virtual world is defined in advance. Users register their interests in classes before participating in the virtual world and only receive messages from their registered classes and subclasses. This approach was adopted by the U.S. Department of Defence [DoD 1998]. The drawback of class-based filtering is that it does not work well in virtual worlds where the interest sets of users change frequently.

3.1.3. Interest Management Implementations. There are many approaches to implementing interest filtering. One is to deploy central interest management and filtering, which is usually combined with unicast to produces a customized stream of data for each user. Example systems include the following.

—*RING [Funkhouser 1995].* All updates are forwarded to a central server, and the server only forwards updates to the hosts with objects that are visible to each other.
—*MASSIVE [Greenhalgh 1998].* Updates are sent to central managers. Viewers must inform the central managers of which subset of the data they are interested in. The managers then decide what data to forward to each node based on the selection criteria.

Interest filtering with unicast communication incurs an additional network cost when a significant number of participants are interested in the same piece of information; hence, multicasting is often used in conjunction with interest filtering to reduce redundant network traffic. Updates are transmitted to multicast groups, and participating processes must join or leave a group to start or stop receiving updates from that group. Either IP multicast or application-layer multicast [Banerjee et al. 2002] could be applied. The key challenge is to partition the available updates among a set of multicast groups and strike a balance between the granularity of update partitioning and multicast grouping.

Group per object. In this approach, each object is assigned a multicast address. An object then subscribes to the objects located within its focus and their multicast groups. This approach can be extended to assign multiple multicast group addresses to an

object, with each multicast group providing a different level of detail (more discussions in Section 3.4).

To support group-per-object, a virtual world needs to provide a way for a subscriber to learn about the objects located nearby and their multicast addresses (content-addressable communication). Some systems provide an object directory service that tracks the current state of objects, for example, the beacon servers proposed in Spline [Barrus et al. 1996]. A beacon server exists for each region in a virtual world and has a designated multicast address for receiving information about objects in the region. The beacon servers also share a well-known multicast group for locating the appropriate beacon server for a particular region.

The most significant drawback with this approach is that one multicast group per object is costly in large virtual worlds. It consumes a large number of multicast addresses and creates significant overhead on participating machines.

Group-per-region. In this approach, a virtual world is divided into regions (also called grids), and each region is assigned a multicast address. As discussed in Section 3.2.1, hexagons, triangles, and irregular sizes have been explored in different systems. Each object transmits its updates to groups corresponding to regions that cover its current location. A subscriber (e.g., the viewer process of a user) joins multicast groups that its area of interest intersects with. Thus, the subscribed multicast groups change as a user's viewpoint moves around. In general, the filtering efficiency depends on the region size. If a region is too small, a host has to subscribe to too many multicast groups. If a region is too large, a host may receive many updates from entities outside its area of interest.

Hybrid aggregation. The goal behind a hybrid approach is to strike a balance between fine-grained partitioning of updates and multicast grouping. On one hand, it aims to partition updates into fine-grained multicast groups so that participants can tune their subscriptions based on local interests to better approximate their areas of interest. On the other hand, it ensures that the partitioning is not too fine-grained such that update transmission degenerates into unicast communication.

One example of hybrid aggregation is the three-tiered interest management system [Abrams et al. 1998]. The first tier is a group-per-region scheme with dynamically sized regions. The information sent to each region is at a low rate and of low fidelity. The second tier is a group-per-object scheme, allowing hosts to subscribe to each individual object's multicast group for fine-grained level of details. The first two layers are protocol independent, allowing multiple protocols to simultaneously exist within the same simulation environment while using the same underlying filtering mechanism. A third tier implements protocol-dependent filtering, allowing a process to receive only the data from the protocol it needs.

Another example of hybrid aggregation is the Quicksilver Multicast protocol (QSM) [Ostrowski et al. 2008] used in live distributed object [Ostrowski et al. 2007], where the design focuses on enterprise-computing environments (shared high-speed LAN) to leverage IP multicasting. It is based on the observation that by using a group-by-object scheme, multicast groups often overlap in regular ways (e.g., hierarchically), because objects related to common topics are often used by people with common interests. As such, objects are aggregated into regions that are carefully divided based on receiver interests, and each region is assigned a multicast address. Figure 4 illustrates the concepts of groups and regions in Quicksilver. There are multiple processes in the system with copies of objects A, B, and C. The processes that have the copies of the same object belong to the same group, shown as Groups A, B, and C in the figure. A region is a set of processeses of members of the same groups. In Figure 4, there are two regions. Region 1 contains processes a, b, c, and d that are all members of Group A, B,

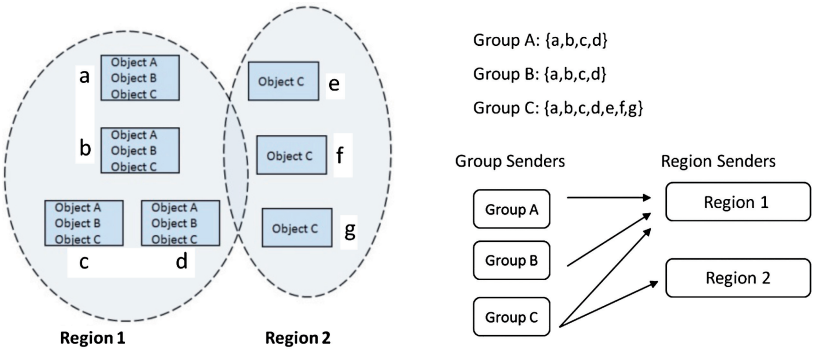


Fig. 4. Illustration of groups, regions, and multicasting to groups in Quicksilver.

and C, while Region 2 contains processes e, f, and g that are only members of Group C. By aggregating groups into regions, it is shown that usually there are much fewer regions than groups, and each process only needs to subscribe to a limited number of regions. For instance, assuming the Zipf popularity model of objects, with 250 to 2,000 processes each subscribing to 10% of 1,000 to 10,000 groups, a process on average belongs to four to ten regions [Ostrowski et al. 2008]. Multicasting to a group is done by transmitting the message to each region the group spans over. As shown in Figure 4, messages for Group A and Group B will be forwarded to the region sender of Region 1 for multicasting to processes in that region, while messages for Group C will be forwarded to the regions senders of Regions 1 and 2 for multicasting into both regions.

3.2. Levels of Detail

By exploiting the inherent limitations of human perception (e.g., inability to perceive high detail at a distance), we can reduce the number or quality of the updates without reducing or only marginally reducing their perceived quality. Information can be provided at multiple levels of detail or at different updating rates. Only users whose viewpoints are located near an object need to receive high-detail information of the object. In addition, updates of remote objects could be delivered at a lower frequency without noticeably affecting a user's experience.

Multi-channel architecture. To meet the needs of different viewer constituencies, each object can transmit updates via multiple independent channels, each providing information at a different level of detail and at a different frequency. A viewer then only subscribes to the channel providing the required level of detail. Different levels of reliability can also be provided to different channels. For instance, a channel with low update frequency may require a more reliable multicast scheme so that receivers do not keep stale information for extended periods of time.

There is a trade-off in deciding how many channels to provide for an entity. If there are too many channels, the management overhead associated with supporting such fine-grained quality levels can mitigate the benefits of having different service levels. It is proposed in Singhal [1997] that typically an object only needs three channels to support far-range, mid-range, and near-range viewers.

—*Rigid-body channel.* This channel demands the least network bandwidth and processor computation for remote viewers. The source transmits enough information to allow receivers to represent the object as a rigid body. This channel provides three types of updates: position, orientation, and significant structure changes.

—*Approximate-body channel.* This channel provides more position and orientation information to remote viewers. It also enables remote viewers to render a rough approximation of the object's dynamic structure, such as the spinning blades of a propeller airplane. The type of structural information is object specific.
—*Full-body channel.* Such a channel provides the highest level of detail about an object's dynamic position, orientation, and structure. It imposes the highest bandwidth and computation.

Aggregated views. Aggregation is another way to explore different levels of detail for supporting high fidelity and low fidelity views, where aggregated views are provided first and may be unfolded at a later time.

In the Paradise project [Singhal and Cheriton 1996], projection aggregation is applied in which each projection aggregation includes objects from a single organization (e.g., a tank that consists of wheels, its body, treads, turret, etc.) located within a single region. Projection updates contain summary information of its objects, such as the number of objects, a single position point summarizing the location of those objects, and the distribution of the objects around that point. On the other hand, a projection is simply an object that transmits summary of one or more subprojections, where the subprojections can be introduced later, and only those viewers requiring that detail need to be aware of it.

A crowd scheme is introduced in Benford et al. [1997] to provide an abstraction of a groups of objects which allows them to be treated as a whole in some cases (e.g., remote entities) but as individuals in other cases (e.g., for interaction among themselves). The scheme was implemented and demonstrated in an arena application.

3.3. Reducing Network Traffic by Taking an Object-Oriented View

In the previous discussions, representation of a virtual world system is described as a set of state variables that represent an instantaneous system state [Singhal and Zyda 1999]. Temporal changes are expressed as a sequence of updates to state variables which need to be broadcasted to participating processes. This traditional view treats consistency as a property of the shared state, and each user's display is derived and rendered from a snapshot of the shared state.

Croquet,[7] in contrast, takes an object-oriented view of the world's state and operation. Croquet is a peer-to-peer system, where each participating process (called an island or replicated island) simulates the behaviors of the objects it hosts. Instead of viewing each object as a set of variables with certain values, it views objects as identities that have behaviors [Smith et al. 2003]. The variables of an object are encapsulated inside the object and invisible outside. Objects send messages to other objects to trigger actions and affect behaviors. Hence, the messages are not state updates, but rather, action triggers. Different objects, or copies of the same object on different islands, can implement different strategies for computing their behaviors (i.e., objects may adapt their behaviors to cope with heterogeneous hardware, delay variations, etc.). One advantage of this model is that in maintaining consistency, the islands do not have to synchronize the detailed values of object variables, but rather, they only need to send messages to coordinate the execution of objects so that all behaviors that can have a visible effect are completed in time and provide the same visible effect to users. However, Croquet's model may have to depend on precise timing and precise message input-output mapping to achieve consistency.

[7]`http://atsosxdev.doit.wisc.edu/croquet2/index.html`.

4. CASE STUDIES

In this section, we review several state-of-the-art virtual worlds and analyze the consistency models and methods of state update dissemination used in each system.

4.1. Second Life

Second Life [8] is one of the most widely publicized virtual worlds. Since its opening to the public in 2003, it has grown rapidly and had more than 13 million registered users as of 2008 [Virtual Worlds News 2008]. Similar to many online gaming systems, Second Life adopts a client-server architecture. There are two primary components: simulators and viewers. Simulators are hosted on servers owned and operated by Linden Lab, while viewers are run on user machines. The virtual world in Second Life is divided into many 256×256 square meter regions. Each region is managed by and only by a simulator process. This is the only process that has an authoritative copy of the objects in the region and applies all operations to the objects, including physics simulation, script execution, management of client connection, and manipulation of objects on behalf of clients. That is, the simulator is solely responsible for updating the state in the region and disseminating the updates.

To view or interact with a Second Life region, a user's viewer connects with the region's simulator. A user can interact with a Second Life object in three ways.

—*Passive interaction.* A user can direct his avatar to bump into an object or chat within audible range of an object. In this case, an update request is sent to the simulator, and the interaction is wholly calculated on the simulator. The results of the interaction are sent to viewers as update messages. Meanwhile, a viewer applies dead reckoning to smooth motions before it receives the updates.
—*Direct interaction.* A user can touch or sit on an object or send a network message (email, HTTP, or XML-RPC) to the object. Again, in this case, an update request is sent to the simulator, the interaction is calculated on the simulator, and updates are sent to viewers.
—*Direct object editing.* If a user has appropriate permissions (such as being an owner of an object), he can modify the object directly, including changing its position and orientation, size, shape, and color. These modifications are immediately applied locally, and an update message is sent to the simulator. If the changes are accepted by the simulator, it is responsible for updating the new object state to all viewers.

Consistency maintenance. In Second Life, since no two simulators share the same region or portions of a region, there is no state shared between simulators. Consistency maintenance is mainly concerned with synchronizing the copies at viewers with the authoritative copies in simulators.

As previously discussed, in Second Life, to enact an operation issued by a user, a request is sent to the simulator hosting that region. Hence, all operations on an entity are arbitrated by the simulator, and the simulator is able to enforce a total order of the operations. In addition, when a user is editing an object, the object is "frozen" by the simulator and cannot be modified by others. This freezing of the object allows the server to clearly delineate between operations on the object that happen before and after the object is modified and maintains a total order of the operations. As a result, sequential consistency is effectively supported in the system.

On the other hand, update messages are not time-stamped. That is, there is no notion of the time or deadline that an update needs to be applied. Updates are processed as they are received, and there is no notion of "late updates". Hence, Second Life does

[8] `http://secondlife.com/`.

not have a mechanism to support timed sequential consistency and only supports sequential consistency.

Update dissemination. A focus-nimbus type of interest management is applied in Second Life. Updates to a viewer are filtered based on the viewer's interests, which are a function of the avatar's position, the size of the object that has generated an update, the position where the update happens, and the viewer's view frustum.

Each viewer constantly sends messages to the simulator to update its camera position (view point) and orientation from which the simulator will calculate the view frustum of the viewer. Periodically, the simulator determines if the state of the region has been updated and generates updates. For each update and for each viewer, the server will determine if the viewer would be interested in the update, and if so, unicasts the update to the viewer. Update messages are sent via UDP packets in a reliable way (viewers need to ACK), but packets could arrive out of order.

4.2. Eve Online

Eve Online[9] is a massively multiplayer online role-playing game (MMORPG) in a science fiction space setting. As of 2008, it has more than 300,000 players [Guðjónsson 2008]. Activities in Eve Online happen in both physical space and social space. In the physical space, players pilot customizable ships to participate in battles through a universe comprising thousands of solar systems. Social space is decomposed into alliances, corporations, gangs, market regions, chat groups, and so forth [Brandt 2005]. In this section, we focus our discussion on state melding in the physical space.

Consistency maintenance. Eve Online (or Eve in short) is also based on the client-server architecture. It maintains a single copy of its game universe, which consists of thousands of connected solar systems on hundreds of blade servers. Each solar system is loaded as a server process onto any of the servers, with some high-load systems being given a server all to themselves and many low-load systems being combined and run together.

Eve uses a technology called *causality bubbles* to scale simulations and synchronize operations [Guðjónsson 2008]. The physics model in Eve is relatively simple: ships are spheres rolling around in vector fields. To scale simulations, Eve dynamically partitions space into causality bubbles, where each bubble is the smallest box that contains all the spheres that are within range of each other and could potentially interact. The space partitioning is done by running a set of differential equations continuously which takes into account the acceleration of every ship in a solar system in order to determine which ships can move within range of each other [Guðjónsson 2008]. Figure 5(a) shows examples of the causality bubbles.

For a solar system, the server process solves the set of differential equations for the entire solar system. To provide low response time, each client process runs some complex dead-reckoning algorithm which solves the set of differential equations for the causality bubble it belongs to. The dead-reckoning algorithm on clients relies on correct timestamps, and the client and server processes are time-synched together.

The server process is the central authority that enforces a total order of the operations and holds the authoritative copy of the system's state. To synchronize the state on each client, the server process regularly sends updates to the initial conditions for a time step. A client process can work its way backwards to adjust for changes in initial conditions and derive how they affect the current state. In this sense, it can be said that Eve supports timed sequential consistency.

[9]`http://www.eveonline.com/`.

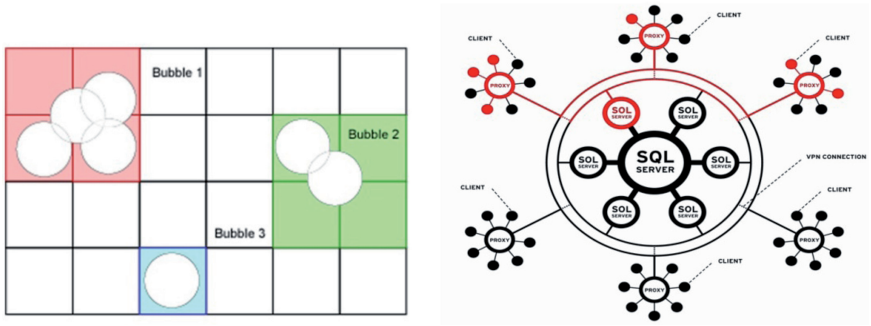


Fig. 5. (a) Causality bubbles; (b) Eve online architecture [Guðjónsson 2008].

Update dissemination. Unlike Second Life and many other online game systems, Eve adopts a different architecture for update dissemination, as shown in Figure 5(b). The solar system servers (SOL servers) are responsible for simulating and evolving the solar systems. SOL servers are all connected to the SQL server—the main database—to persist the world's state. Clients are not connected to SOL servers directly. Rather, they are connected through proxy servers which keep track of which SOL server a player is on. State updates are therefore multicast to clients via proxies using TCP/IP [Brandt 2005]. It is a simple form of application-layer multicast [Banerjee et al. 2002]. Effectively, the load of update dissemination is off-loaded from SOL servers and distributed among proxy servers.

4.3. Project Darkstar

Project Darkstar [Waldo 2008] is a research effort attempting to build a server-side infrastructure for virtual worlds, especially massively multiplayer online games, with the ability to support dynamic allocation of hardware to scale server operations. RedDwarf[10] is a fork of Darkstar, continuing the research and development effort since Feburary 2010. Unlike other online game systems which are built for the specific gameplay of their games, Darkstar strives to be a well-designed server infrastructure that can be used to build different specific game systems.

Traditionally, virtual world systems follow a simulator-centric architecture [Liu et al. 2010] in which each server process (a simulator) owns both the state (the data) and the simulation work of a unique space (e.g., regions in Second Life, solar systems in Eve Online, and shards in World of Warcraft). The data and the simulation operations in the space are tightly coupled and physically co-located in the same simulator. This simulator-centric architecture has been observed to be a critical scalability bottleneck [Liu and Bowman 2010; Liu et al. 2010] and does not have the flexibility to dynamically allocate hardware to scale operations.

Darkstar took a different approach and broke away from the simulator-centric architecture. Instead of having a set of simulators, each hosting a space, and all the data and operations in that space, it views a virtual world as a collection of tasks that operate on the data (the world state). Figure 6 presents the high-level architecture of a Darkstar system. In general, Darkstar models virtual world operations as an event-based system in which events (e.g., a client's input) trigger tasks, and tasks simulate world operations and evolve the world state. When an input from a client is received by a Darkstar game server (a machine that runs the Darkstar stack and the game logic), a

[10]`http://www.reddwarfserver.org/`.

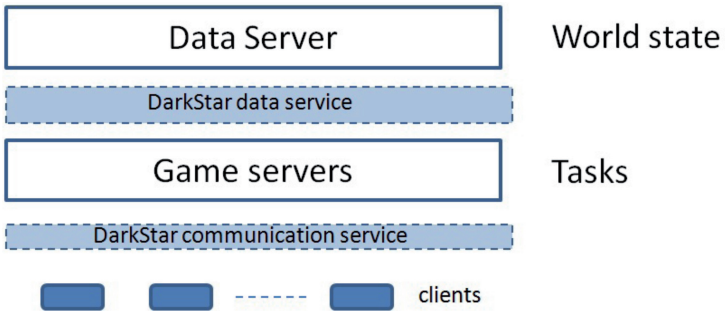


Fig. 6. High-level architecture of Darkstar system.

task is set off in response to that event. The world state (the data) is stored on the data store, where a Berkeley Database system is used to achieve fast access to persistently stored objects. By requiring that all data be kept in the data store and all tasks access data through the data service, tasks become portable among game servers, since the data store can be accessed by any game server. As a result, tasks can be dynamically assigned to different game servers to scale the system's operations by adding or removing hardware.

Consistency maintenance. In a virtual world built upon Darkstar, there are two levels of consistency controls. One level is to maintain the state on client machines to be consistent with the authoritative copy on the servers. The basic consistency control provided in Darkstar is similar to that of Second Life in that the servers impose an order on user inputs and disseminate state updates to clients. Since servers have the authoritative copy of the state, the updates will overwrite any inconsistent state on clients. This basic consistency model effectively achieves sequential consistency.

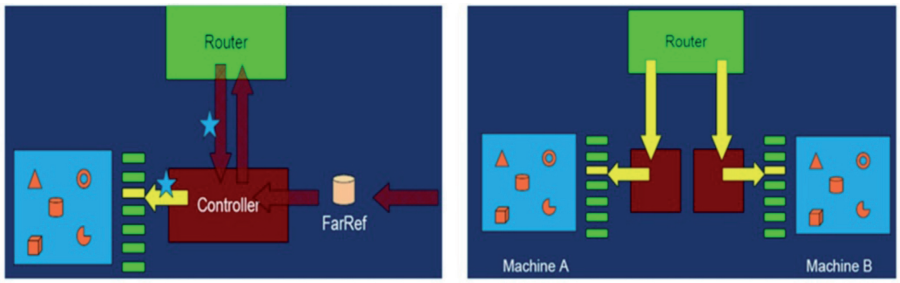
The second level of consistency control in Darkstar is to synchronize the state among the servers. In general, each Darkstar server hosts a set of tasks that access the world state through the data service. But to reduce data access delays, each game server also maintains its own data cache to store local copies of recently used objects [Blackman and Waldo 2009]. Hence, different game servers might share copies of the same object. To maintain consistency of each server's local copy, a task running on a server is wrapped in a transaction. If a task tries to change data that is being modified by another concurrent task, the data service will detect the conflict, and one of the tasks will be aborted and rescheduled. This essentially achieves serializability for operations across game servers.

Update dissemination. Currently, there is no disseminate protocol implemented in Darkstar. It does implement a thin communication API for clients to exchange messages with servers, and it is possible that a pub/sub-system can be built on top of it. A design goal of Darkstar is to enable clients to migrate among servers so that clients that access similar sets of data could be co-located on the same server. This helps balance the load as well as reduce access delays for clients.

4.4. Croquet

Croquet[11] is an open-source software development environment and is focused on context-based collaborative applications. Qwaq Forum™[12] is built based on Croquet

[11] http://atsosdev.doit.wisc.edu/croquet2.html.
[12] http://teleplace.com/company/qwaq_is_now_telepalce.php.



(a) Router adds timestamps to all external messages.

(b) Router sends heartbeat messages to move time forward.

Fig. 7. Timing in Croquet [Smith et al. 2005].

and focused on business collaboration usages. Croquet adopts a decentralized peer-to-peer architecture where each participating host has a replicated copy of an object and applies operations to the object to simulate and evolve the world. Croquet objects reside in islands (object containers), and the islands are replicated across hosts.

Consistency maintenance. Sequential consistency is achieved in a Croquet system. For a set of replicated islands, there is one router that acts as the clock and time-stamps messages, as shown in Figure 7. The router is a centralized entity, and the same set of replicated islands communicate with the same router. By time-stamping messages, the router effectively places a total order on the messages. Thus, all replicated islands execute the messages in the same order and produce the same visible effect to users.

Update dissemination. As discussed in Section 3.1, Croquet defines objects as entities that have behaviors, and messages affect these behaviors. State variables are invisible outside the object. An object is a time-independent programmed mapping from a set of input messages to a set of output messages, where the programming of the object just defines the mapping. Hence, in Croquet, only messages for triggering behaviors need to be exchanged, and objects are free to implement a wide variety of strategies for computing their behaviors. This could potentially reduce network traffic by avoiding delivery of detailed updates

Croquet uses a virtual overlay structure to implement some-to-some (one-to-many, many-to-one, many-to-many) deliveries to facilitate message dissemination [Reed 2005].

5. DISCUSSIONS

Maintaining a consistent view for a virtual world and updating users with unnoticeable delay is a great challenge, especially when scaling virtual worlds far beyond their current capability in supporting a large number of concurrent users, complex scenes, high fidelity of interactions, and long viewing distances. Besides applying appropriate consistency models and optimized update disseminations, it also requires redesign of system architectures or other innovative approaches. In this section, we discuss the challenges and potential solutions for state melding in large-scale virtual worlds.

5.1. Addressing Scalability in State Melding

Most virtual worlds, including online games, have adopted the client-server architecture, where the role for the server is two-fold. First, it enforces a total order of operations, resolves concurrent access conflicts, and persists the world's state, hence providing users with a consistent context to interact in real time. Second, it holds the

authoritative copy of the world's state and is the arbiter of the true state for users. It is used both to discourage cheating and detect cheating. The client-server architecture, however, could greatly limit the scalability of the system if not designed carefully. Most of today's virtual worlds have applied a simulator-centric architecture [Liu et al. 2010], where each server process manages a portion of the space in the world and owns all the work in that space: managing the data structure (the state), running simulations (the operations that alter and evolve the state), and disseminating state updates to all connected clients. In practice, a system using this architecture, such as Second Life and World of Warcraft, has difficulty in supporting more than 100 users interacting with each other concurrently (where each user is able to observe and respond to any other users' actions) [Gupta et al. 2009]. Such systems use sharding or region partitioning to partition users into separate, isolated groups. Users can only interact with others that are also in the same shard or region. Eve Online applies specific optimizations based on deterministic physics models (where ships are modeled as spheres rolling around) to scale a solar system's operation and the number of concurrent users in the same solar system. Such specific optimizations, however, cannot be easily extended to general operations in virtual worlds. On the other hand, an architecture improvement in Eve Online, that is, off-loading the client management work to proxy servers, mitigates the communication bottlenecks between servers and clients and can be extended to other systems.

To address the scalability limitations and scale virtual worlds beyond their current capabilities, several approaches have been proposed to break the simulator-centric architecture: scalable system architecture [Horn et al. 2009, Liu et al. 2010], scalable data storage [Waldo 2008], scalable message exchanging [Horn et al. 2010], and scalable synchronization protocol [Gupta et al. 2009].

Distributed Scene Graph (DSG) is a new server architecture proposed in Liu et al. [2010] for virtual worlds based on client-server models. Here, scene graph is a general term that refers to the scene in the virtual world. More specifically, scene graph refers to the data structure that stores the state of the world, including meta-data of the space (e.g., sun's position, ocean level, etc.), all the objects in world, and the current values of each object's properties, such as shape, scale, position, orientation, etc.

DSG shares a similar spirit with Darkstar in that it views operations in virtual worlds as a collection of the scene and the actors operating on the scene. Each actor observes a portion of the scene and applies a specific set of operations to the objects in this portion of the scene. For instance, physical engine, script engine, and client managers (receiving inputs from clients and manipulating objects on their behalf and disseminates updates to them) are examples of actors. These actors operate on the objects through the scene interface and thus become portable, which in turn enables them to run on separate hardware and scale independently. For efficient data access, each actor also maintains a local copy of the portion of the scene it operations on. Note that actors could be either computation or communication intensive. As illustrated in Figure 8, for computation-intensive actors, DSG enables application of dynamic load balancing to scale their operations. For communication-intensive actors (e.g., the client managers), DSG enables them to be detached from other actors and run on separate hardware that is provisioned for high-speed networking. A DSG prototype based on OpenSimulator[13] has demonstrated the ability to support more than 1,000 concurrent client connections, each having an avatar interacting with each other concurrently in the same space [Lake et al. 2010].

However, disaggregating actors brings back the challenges that were bypassed in the simulator-centric architecture. In the simulator-centric architecture, since a simulator

[13]http://opensimulator.org/wiki/Main_Page.

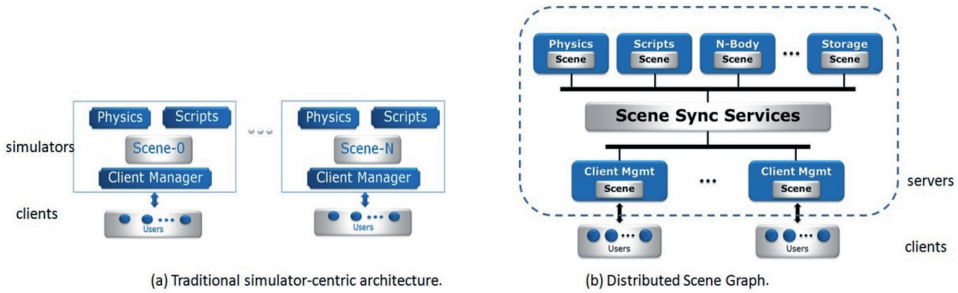


Fig. 8. Simulator-centric architecture versus Distributed Scene Graph.

does not share data with others, it is the sole arbitrator of user inputs and dictates the state of the space it hosts. In DSG, however, actors need to synchronize their replicated state of the world, since two actors may operate on the same portion of the scene. In the DSG prototype, a timestamp-based synchronization service has been implemented to provide the synchronization service, assuming the servers' clocks are synchronized with very limited clock drifting (e.g., using NTP). When an actor modifies any property of an object, it time-stamps the updated value and propagates the update with the timestamp. For other actors, they compare the timestamp of the update with that in their local copy and apply the update if it has a newer timestamp; otherwise, the update is discarded. For update conflicts, (two updates on the same property having the same timestamps), a policy is implemented to be the tie-breaker, and all actors implement the same policy. Usually, different actors perform write operations at different frequencies, and they each tend to only modify a subset of object properties (e.g., physics engine modifies only physics properties). Hence, in practice, write conflicts happen very infrequently.

For maintaining consistency of the copies on clients (so that they are also consistent with those on servers), DSG follows the same model as in Second Life and Eve Online: the collective state on servers holds the authoritative copy of the system's state. To synchronize the state on each client, servers (the client managers in particular) regularly send out updates. A client may run algorithms, such as dead reckoning, to advance its local state for fast responsiveness but needs to correct its local state upon receiving updates from servers and detecting that its local state is inconsistent.

5.2. Addressing the Consistency-Responsiveness Trade-off

Another big challenge in state melding is to reduce the response delay while maintaining consistent views across users so as to improve interaction fidelity and provide truly immersive experiences. However, as discussed before, optimization of response time and avoidance of short-term inconsistency are conflicting goals due to network delays.

To reduce network delays in disseminating updates, one approach is to extend the two-tier client-server architecture into three tiers. For example, based on the DSG architecture just discussed, a middle layer can be added in between clients and servers to assist in efficiently customizing, aggregating, and disseminating updates to clients. This middle layer is referred to as the *reduction pipeline*. It could be an extension of the client management, as shown in Figure 8, and include operations of client management, interest management, and detail reduction (reduction of frequency and/or fidelity of updates) and aggregations. It is similar to the role of Akamai servers[14] in providing content-delivering services for Web applications. Servers in the reduction pipeline can

[14]http://www.akamai.com/.

be strategically placed (e.g., geographically closer to a group of clients) so as to reduce delays in delivering updates to clients.

There are, however, several challenges in developing reduction pipelines for virtual worlds, as compared to content delivery networks. First, it needs to meet the strong real-time requirements in delivering updates and synchronizing the states. Second, each client in a virtual world usually has a unique viewpoint and hence has its own perspective of the world. That is, the reduction pipeline needs to meet the requirement of real-time visualization for many users with customized perspectives. The solution presented in Chaudhuri et al. [2008] is one attempt to address this above challenges in which overlay services are introduced for fast customization and delivery of world views to many concurrent users.

6. CONCLUSIONS

This article presents a comprehensive survey of state melding-related techniques in virtual worlds. The fundamental goal of virtual worlds is to provide users with the illusion that they are all seeing the same things and interacting with each other in a highly connected virtual world. Scalable and efficient state melding implementations are key to achieving this fundamental goal.

There are two main steps in state melding: state consistency maintenance and state update dissemination. In this survey, we reviewed several ultimate consistency models and deadline-based consistency models. In general, deadline-based consistency models are more suitable for virtual world applications due to their better support for real-time interactions. However, in practice, almost no large-scale virtual world has implemented such a model. This may be due to the complexity of the protocols and the lack of understanding of their performance in large-scale systems.

To provide desired consistency support with acceptable complexity, one approach may be to identify and categorize objects and operations with different consistency requirements and to apply different consistency models to different categories instead of applying the same consistency model across the system.

There are still many open research questions in developing state melding technologies, especially when scaling virtual worlds far beyond their current capabilities in terms of concurrent users, scene complexity, fidelity of interaction, and viewing distances. There are several promising research directions in addressing the big challenge. In particular, research has shown that it is essential to break the simulator-centric architecture and decouple the state with the operations that evolve the state. By doing so, different operations could run on different hardware and be optimized and load-balanced independently. In addition, decoupling data and operations enables introducing a reduction pipeline service between servers and clients for customizing and accelerating update delivery for clients and for addressing the conflicting goal of maintaining a consistent view while providing fast responsiveness to user inputs.

ACKNOWLEDGMENTS

The authors thank John Hurliman, Robert Adams, Dan Lake, and John D. Miller from Intel, Andreas Raab from Qwaq Forums, DaviðBrandt from NARC (formerly with Eve Online), Jeff Kesselman from Blue Fang (formerly with Project Darkstar), and the reviewers for their valuable inputs.

REFERENCES

ABRAMS, H., WATSEN, K., AND ZYDA M. 1998. Three-tiered interest management for large-scale virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST'98)*. ACM, New York, NY, 125–129.

AHAMAD, M., BURNS, J. E., HUTTO, P. W., AND NEIGER, G. 1991. Causal memory. In *Proceedings of the 5th International Workshop on Distributed Algorithms (WDAG'91)*, S. Toueg, P. G. Spirakis, and L. M. Kirousis, Eds., Lecture Note in Computer Science, vol. 579, Springer-Verlag, Berlin, 9–30.

BANERJEE, S., BHATTACHARJEE, B., AND KOMMAREDDY, C. 2002. Scalable application layer multicast. *SIGCOMM Comput. Commun. Rev.* 32, 4, 205–217.

BARRUS, J. W., WATERS, R. C., AND ANDERSON, D. B. 1996. Locales and beacons: Efficient and precise support for large multi-user virtual environments. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, 204–213.

BENFORD, S. AND FAHLÉN, L. 1993. A spatial model of interaction in large virtual environments. In *Proceedings of the 3rd European Conference on Computer-Supported Cooperative Work (ECSCW'93)*, G. de Michelis, C. Simone, and K. Schmidt, Eds. 109–124.

BENFORD, S., GREENHALGH, C., AND LLOYD, D. 1997. Crowded collaborative virtual environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'97)*. ACM, New York, NY, 59–66.

BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. 1987. *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Boston, MA.

BLACKMAN, T. AND WALDO, J. 2009. Scalable data storage in Project Darkstar. Tech. rep. UMI order number: SERIES13103. Sun Microsystems, Inc.

BOUILLOT, N. 2004. The auditory consistency in distributed music performance: A conductor based synchronization. *Inf. Sci. Decision Making 13*, 129–137.

BOUILLOT, N. AND GRESSIER-SOUDAN, E. 2004. Consistency models for distributed interactive multimedia applications. *SIGOPS Operat. Syst. Rev. 38*, 4, 20–32.

BOULANGER, J., KIENZLE, J., AND VERBRUGGE, C. 2006. Comparing interest management algorithms for massively multiplayer games. In *Proceedings of the 5th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames'06)*. ACM, New York, NY.

BRANDT, D. H. 2005. Scaling EVE Online: Under the hood of the network layer. In *Proceedings of the 4th Workshop on Network & System Support for Games*.

CHAUDHURI, S., HORN, D., HANRAHAN, P., AND KOLTUN, V. 2008. Distributed rendering of virtual worlds. Tech. Rep., CSTR 2008-02, Stanford University, Stanford, CA.

CRONIN, E., FILSTRUP, B., KURC, A. R., AND JAMIN, S. 2002. An efficient synchronization mechanism for mirrored game architectures. In *Proceedings of the 1st Workshop on Network and System Support for Games (NetGames'02)*. ACM, New York, NY, 67–73.

DELANEY, D., WARD, T., AND MCLOONE, S. 2006. On consistency and network latency in distributed interactive applications: A survey—part I. *Presence: Teleoper. Virtual Environ. 15*, 2, 218–234.

DoD. 1998. U.S. Department of Defense, High Level Architecture Interface Specification, Version 1.3.

ELLIS, C. A. AND GIBBS, S. J. 1989. Concurrency control in groupware systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'89)*, J. Clifford, B. Lindsay, and D. Maier, Eds. ACM, New York, NY, 399–407.

FIEDLER, S., WALLNER, M., AND WEBER, M. 2002. A communication architecture for massive multiplayer games. In *Proceedings of the 1st Workshop on Network and System Support for Games (NetGames'02)*. ACM, New York, NY, 14–22.

FUNKHOUSER, T. A. 1995. RING: A client-server system for multi-user virtual environments. In *Proceedings of the Symposium on Interactive 3D Graphics (I3D'95)*. ACM, New York, NY.

FUJIMOTO, R. M. 1999. Parallel and distributed simulation. In *Proceedings of the Winter Simulation Conference*. vol. 1, 122–131.

FUJIMOTO, R. M. 2003. Distributed simulation systems. In *Proceedings of the Winter Simulation Conference*. vol. 1, 124–134.

GAUTIER, L. AND DIOT, C. 1998. Design and evaluation of MiMaze, a multi-player game on the Internet. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS'98)*. IEEE Computer Society, Los Alamitos, CA.

GREENHALGH C. 1998. Awareness-based communication management in the MASSIVE systems. *Distrib. Syst. Eng. 5*, 3, 129–137.

GREENHALGH, C., PURBRICK, J., AND SNOWDON, D. 2000. Inside MASSIVE-3: Flexible support for data consistency and world structuring. In *Proceedings of the 3rd International Conference on Collaborative Virtual Environments (CVE'00)*, E. Churchill and M. Reddy, Eds. ACM, New York, NY, 119–127.

GUÐJÓNSSON, H. F. 2008. The server technology of EVE Online: How to cope with 300,000 players on one server. In *Proceedings of the Austin Game Developers Conference (Austin GDC'08)*.

GUPTA, N., DEMERS, A., GEHRKE, J., UNTERBRUNNER, P., AND WHITE, W. 2009. Scalability for virtual worlds. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'09)*. IEEE Computer Society, Los Alomitos, CA, 1311–1314.

HORN, D., CHESLACK-POSTAVA, E., AZIM, T., FREEDMAN, M. J., AND LEVIS, P. 2009. Scaling virtual worlds with a physical metaphor. *IEEE Pervasive Comput. 8*, 3, 50–54.

HORN, D., CHESLACK-POSTAVA, E., MISTREE, B., AZIM, T., TERRACE, J., FREEDMAN, M. J., AND LEVIS, P. 2010. To infinity and not beyond: Scaling communication in virtual worlds with Meru. Stanford Computer Science Tech. rep. CSTR 2010-01.

IEEE. 1993. IEEE standard for information technology—protocols for distributed simulation applications: Entity information and interaction. *IEEE Standard 1278–1993*. IEEE Computer Society, New York, NY.

JEFFERSON, D. R. 1985. Virtual time. *ACM Trans. Prog. Lang. Syst. 7*, 3, 404–425.

JOHNSON, D. B. AND MALTZ, D. A. 1996. Dynamic source routing in ad hoc wireless networks. *Mob. Comput. 353*, 153–181.

KANAWATI, R. 1997. LICRA: A replicated-data management algorithm for distributed synchronous groupware application. *Parallel Comput. 22*, 13, 1733–1746.

KEATING, T. 2007. Dupes, speed hacks and black holes: How players cheat in MMOs. In *Proceedings of the Austin Game Developers Conference (Austin GDC'2007)*.

LAKE, D., BOWMAN, M., AND LIU, H. 2010. Distributed scene graph to enable thousands of interacting users in a virtual environment. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games (NetGames'10)*.

LAMPORT, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7, 558–565.

LAMPORT, L. 1979. How to make a multiprocessor computer that correctly executes multi-process programs. *IEEE Trans. Comput. C28*, 9, 690–691.

LIU, H. AND BOWMAN, M. 2010. Scale virtual worlds through dynamic load balancing. In *Proceedings of the 14th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT'10)*. IEEE Computer Society, Los Alamitos, CA, 43–52.

LIU, H., BOWMAN, M., ADAMS, R., HURLIMAN, J., AND LAKE, D. 2010. Scaling virtual worlds: Simulation requirements and challenges. In *Proceedings of the Winter Simulation Conference*, 778–790.

LUI, J. C. S., SO, O. K. Y., AND TAM, P. T. S. 1999. Deriving communication sub-graph and optimal synchronizing interval for a distributed virtual environment system. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*.

MACEDONIA, M. R., ZYDA, M. J., PRATT, D. R., BRUTZMAN, D. P., AND BARHAM, P. T. 1995. Exploiting reality with multicast groups. *IEEE Comput. Graphics Appl. 15*, 5, 38–45.

MAUVE, M. 1999. TeCo3D: A 3D telecooperation application based on VRML and Java. In *Proceedings of the SPIE Multimedia Computing and Networking (MMCN)*.

MAUVE, M. 2000. Consistency in replicated continuous interactive media. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'00)*. ACM, New York, NY, 181–190.

MOSS, E. 1985. Nested transactions: An approach to reliable distributed computing. The MIT Press, Cambridge, MA, 31–38.

NEWMAN-WOLFE, R. E., WEBB, M. L., AND MONTES, M. 1992. Implicit locking in the ensemble concurrent object-oriented graphics editor. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'92)*. ACM, New York, NY, 265–272.

OSTROWSKI, K., BIRMAN, K., AND DOLEV, D. 2007. Live distributed objects: Enabling the active Web. *IEEE Internet Comput. 11*, 6, 72–78.

OSTROWSKI, K., BIRMAN, K., AND DOLEV, D. 2008. Quicksilver Scalable Multicast (QSM). In *Proceedings of the 7th IEEE International Symposium on Network Computing and Applications (NCA'08)*. IEEE Computer Society, Los Alamitos, CA, 9–18.

QIN, X. 2002. Delayed consistency model for distributed interactive systems with real-time continuous dedia. *J. Softw. 13*, 6, 1029–1039.

RAMJEE, R., KUROSE, J., TOWSLEY, D., AND SCHULZRINNE, H. 1994. Adaptive playout mechanisms for packetized audio applications in widearea networks. In *Proceedings of the Conference on Computer Communications (InfoCom'94)*, 680–688.

RAYNAL, M. AND SCHIPER, A. 1995. From causal consistency to sequential consistency in shared memory systems. In *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, P. S. Thiagarajan, Ed. Lecture Notes in Compouter Science, vol. 1026, Springer-Verlag, Berlin, 180–194.

RAYNAL, M. AND SCHIPER, A. 1996. A suite of formal definitions for consistency criteria in distributed shared memories. In *Proceedings of the ISCA 12th International Conference on Parallel and Distributed Computing (PDCS'96)*.

RAYNAL, M., THIA-KIME, G., AND AHAMAD, M. 1997. From serializable to causal transactions for collaborative applications. In *Proceedings of the 23rd Euromicro Conference: New Frontiers of Information Technology (Euromicro'97)*.

REED, D. P. 2005. TeaTime: Designing the architectural framework for Croquet. `http://atsosxdev.doit.wisc.edu/croquet2/about_croquet/papers.html`.

ROBERTS, D. J. 1996. A predictive real time architecture for multi-user, distributed, virtual reality. Ph.D. dissertation University of Reading.

ROBERTS, D. J., AND SHARKEY, P. M. 1997. Minimising the latency induced by consistency control, within a large scale multi-user distributed virtual reality system. In *Proceedings of the IEEE International Conference on Computational Cybernetics and Simulation.* 4492–4497

SECOND LIFE WIKi. 2008. AWG: State melding exploration. `http://wiki.secondlife.com/wiki/AWG:_state_melding_exploration`.

SINGHAL, S. K. 1997. Effective remote modeling in large-scale distributed simulation and visualization environments. Ph.D. dissertation. Stanford University, Stanford, CA. UMI order No. GAX97-14191.

SINGHAL, S. K., AND CHERITON, D. R. 1996. Using projection aggregations to support scalability in distributed simulation. In *Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS'96)*. IEEE Computer Society, Los Alamitos, CA.

SINGHAL, S. AND ZYDA, M. 1999. *Networked Virtual Environments*. ACM Press/Addison-Wesley, New York, NY.

SMITH, D. A., KAY, A., RAAB, A., AND REED, D. P. 2003. Croquet—A collaboration system architecture. In *Proceedings of the 1st Conference on Creating, Connecting and Collaborating through Computing.*

SMITH, D. A., RAAB, A., REED, D. P., AND KAY, A. 2005. Hedgehog architecture. `http://atsosxdev.doit.wisc.edu/croquet2/about_croquet/papers.html`.

STEINMAN, J. 1993. Breathing time warp. *ACM SIGSIM Simul. Digest, 23*, 1, 109–118.

SUN, C. AND CHEN, D. 2000. A multi-version approach to conflict resolution in distributed groupware systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS'00)*. IEEE Computer Society, Los Alamitos, CA.

TORRES-ROJAS, F. J., AHAMAD, M., AND RAYNAL, M. 1999. Timed consistency for shared distributed objects. In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing (PODC'99)*. ACM, New York, NY, 163–172.

VIRTUAL WORLDS NEWS. 2008. Interview: Strategy analytics' Barry Gilbert—137M virtual worlds users now; 1B by 2017. *Engage Digital*, June 2008.

WALDO, J. 2008. Scaling in games and virtual worlds. *Commun. ACM 51*, 8, 38–44.

WELCH, J. L. 1994. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst. 12*, 2, 91–122.

ZHOU, S., CAI, W., LEE, B., AND TURNER, S. J. 2004. Time-space consistency in large-scale distributed virtual environments. *ACM Trans. Model. Comput. Simul. 14*, 1, 31–47.

Received April 2009; revised December 2010; accepted March 2011