Abstract

A distributed virtual environment (DVE) is a software system that allows users in a network to interact with each other by sharing a common view of their states. As users are geographically distributed over large networks like the internet and the number of users increases, scalability is a key aspect to consider for real-time interaction. Various solutions have been proposed to improve the scalability in DVE systems but they are either focused on only specific aspects or customized to a target application. In this paper, we classify the approaches for improving scalability of DVE into five categories: communication architecture, interest management, concurrency control, data replication, and load distribution. We then propose a scalable network framework for DVEs, ATLAS. Incorporated with our various scalable schemes, ATLAS meets the scalability of a system as a whole. The integration experiences of ATLAS with several virtual reality systems ensure the versatility of the proposed solution.

1 Introduction

A distributed virtual environment (DVE) is a software system that allows real-time interaction among geographically distributed users by providing various levels of sharing in terms of space, presence, time, and so on (Singhal & Zyda, 1999). A shared context among users is achieved by sharing each user’s activities with the rest of the users and often enhanced by replicating the information at each user’s site. However, as users are geographically distributed over large networks such as the internet and the number of users increases, scalability is a key aspect to consider for supporting real-time interaction (Macedonia & Zyda, 1997).

Various attempts have been made to improve scalability in DVEs. One of the key design issues is to reduce message exchanges between users as much as possible without harming the shared context and interactive performance. The most typical approach is filtering unnecessary messages by dividing a virtual world into several regions (Barrus, Waters, & Anderson, 1996; Macedonia, Zyda, Pratt, Brutzman, & Barham, 1995) or localizing the area of interest of users (Benford & Greenhalgh, 1997; Funkhouser, 1995; Hagsand, 1996). However, if filtering is solely done by a server, it burdens the server with a
flood of messages and thus increases network delay. Massively multi-user online role-playing games (MMORPG; Lineage 2, 2004; EverQuest II, 2004; Asheron’s Call 2, 2004) adopt the client/server model, and succeed in terms of scalability. However, they still have the bottleneck problem if lots of users are clumped in a server. To avoid this, many have adopted the peer/peer or peer/server models with multicast support (Barrus et al.; Benford & Greenhalgh; Hagsand; Macedonia et al.). Recently, various approaches to applying the peer/peer architecture to online games also have been devised (Hu & Liao, 2004; Limura, Hazeyama, & Kadobayashi, 2004). Another way to reduce the amount of message exchanges is to replicate the virtual world data at the client from the server. A key concern here is how to efficiently reduce transmission delay of run-time changes in the virtual world data (Capps, 2000; Chim et al., 1998). In addition, replication requires synchronization among replicas in the presence of multiple concurrent updates, which eventually lead to inconsistent views among users (Roberts & Sharkey, 1997). Sometimes scalability issues cannot overcome the limit of physical capability of a server, which brings the requirement of multiple servers (Funkhouser). What we should consider here is how to distribute different loads among servers in order to make the whole system efficient and scalable (Ng, Si, Lau, & Li, 2002). In summary, scalability improvement in DVEs cannot be done in a single aspect but has to be dealt with in various aspects such as communication architecture, interest management, concurrency control, data replication, and load distribution.

Existing systems have proposed various solutions to the scalability problem but their approaches are limited to specific applications and do not satisfy five scalability requirements according to various DVE applications. NPSNET (Macedonia et al., 1995; Capps, McGregor, Brutzman, & Zyda, 2000) is software architecture for large-scale battlefield simulation. MASSIVE (Greenhalgh & Benford, 1995; Benford & Greenhalgh, 1997; Greenhalgh, Purbrick, & Snowden, 2000; Purbrick & Greenhalgh, 2003), and DIVE (Hagsand, 1996; Frécon, Smith, Steed, Stenius, & Stahl, 2001) are developed for a 3D conferencing system that concentrates on interaction among individual users. They attempt to increase scalability by using a spatial filtering mechanism and adopting the peer-to-peer communication architecture. However, their interest management schemes are not appropriate for collaborative engineering applications since they pay little consideration to data replication or concurrency control. PaRADE (Roberts & Sharkey, 1997) attempts to improve interactive performance in concurrency control by using a prediction-based scheme, and applies it to a ballgame application. It, however, is not scalable when many users exist in a virtual world. QUICK (Capps, 2000) only focuses on quality management of dynamic objects distribution. Some systems (Lui & Chan, 2002; Ng et al., 2002; Pekkola et al., 2000) provide various load distribution schemes based on the fundamental region-based interest management in the environment of multiple servers such as online games, but they overlook issues such as concurrency control and data replication. As a framework approach, OpenPING (Okanda & Blair, 2004) is a reflective middleware adapting to dynamic application requirements. While it focuses on flexibility issues in terms of concurrency control, data replication, persistency, consistency, event channeling, and interest management, OpenPING does not consider load distribution which is important in multi-server-based applications. Bamboo (Watsen & Zyda, 1998) is another low level approach and provides application designers with interfaces for device management, networking, graphical user interface, and extensibility. Since it delegates to application developers the specific mechanisms, Bamboo does not support high level functionalities such as interest management, concurrency control, data replication, and load distribution.

In this paper, we propose a scalable network framework for DVEs, ATLAS. ATLAS’ communication architecture provides not only the client/server model, but also a peer/server model in order to adapt to the different requirement of applications. To improve interactive performance, ATLAS’ interest management allows a user in a region to receive update messages with high frequency from only those who have the same interests with him or her, while receiving with low frequency messages from users of different interests. It also pro-
vides a scheme that allows a user to receive update messages from only those users in the neighboring regions who are of high possibility of interaction with him or her (i.e. not all of the users). To resolve conflicts of concurrent updates of objects and grant an ownership to a right user in time, ATLAS provides a prediction-based concurrency control scheme as well as a lock-based one. In the ATLAS prediction-based scheme, the current owner of an object receives ownership requests only from users adjacent to the object, not from all users in the same region, and determines the next owner based on the predicted collision time. When replicating objects from the server at local hosts, ATLAS provides a data management scheme that reduces downloading time by caching and prefetching a subset of objects in a given region that is close to a target user, and has high locality of reference based on the user’s behavior pattern. For efficiently distributing loads among servers, ATLAS’ load distribution management provides a scheme by which an overloaded server selects a set of servers to be involved in load distribution, dynamically adapting to the workload status of other servers.

For supporting various applications, ATLAS introduces an intermediate layer playing the role of routing and transforming messages between ATLAS and applications. We have integrated ATLAS with several applications for ensuring its versatility.

The rest of the paper is organized as follows. In Section 2, we discuss five scalability considerations in large DVE systems. Section 3 presents the scalability solutions in ATLAS, its implementation, and integration experiences with various applications. Conclusion with future work follows in Section 4.

2 Scalability Considerations

There are many issues related to interactive performance in DVEs (Singhal & Zyda, 1999; Capps & Stotts, 1997). In this section, we discuss the five key design issues that should be considered for scalability of DVEs, and argue that these issues cannot be dealt with in a specific manner since scalability requirements vary from application to application.

2.1 Communication Architecture

Since a DVE enables geographically dispersed users to share the same context and to maintain a persistent state by exchanging states of the users, how to reduce communication overhead is a key design consideration. Depending on how the communication between users is coordinated, its architecture can be characterized as follows: client/server, peer/peer, or peer/server.

In the client/server architecture, communication between users has to be done via a server. To interact with other users, each user should transmit messages to the server which then forwards them to other users. This architecture benefits from a simple consistency and security mechanism for which the server is responsible. The server can perform efficient message filtering by forwarding data to only selected receivers according to their interests, for instance. However, this architecture has scalability problems. First, since all the messages among users must pass through the server, they require one additional hop to reach the receivers, which can increase delay in transmission time. Second, when the server fails to operate, all clients connected to the server cannot interact with each other until the server recovers. Finally, as the number of clients connected to the server increases, it may become a performance bottleneck.

In spite of the problems described above, some existing DVE systems adopt the client/server architecture as their communication architecture, since it can efficiently keep the system consistent and be well deployed in the current internet without multicast capability. For scalability, some systems introduce multiple servers to overcome the limit in one server in terms of the number of clients and a single point of failure (Funkhouser, 1995).

In the peer/peer architecture, there is no server involved in the communication between clients. Thus, no single point of failure problem exists. Even if some clients fail to operate, the rest can interact transparently. The clients can interact with each other via a direct communication channel. Since there exists no server maintaining a consistent state, each client host is responsible for keeping its state consistent with others. This imposes a burden on the client. Especially if unicast
communication is used, each client has to maintain as many connections as the number of other clients with which it intends to interact. The more clients exist, the more connections are required. That is, a scalability problem occurs. Multicast communication is used to avoid this problem in the peer/peer architecture where each user maintains only one multicast address assigned to the group that he or she is interested in. While multicast can reduce the communication overhead, fine-grained message filtering such as filtering per user is not possible as in the unicast communication. Another problem of multicast is its slow deployment in the internet due to various reasons (Diot, Levine, Lyles, Kassem, & Balensiefen, 2000): (1) All routers in the existing network infrastructure do not support multicast and replacement cost is high; (2) If a multicast protocol (PIM-SM or CBT) adopts a shared tree, ISPs do not want to have a rendezvous point (RP) because it is a waste of their resources; (3) Due to the poor interoperability with existing services, multicast is difficult to install and manage. For example, multicast addresses are not recognized by most firewalls. Due to the above reasons, the deployment and management cost is higher than the benefit gained from saving bandwidth. In addition to linking multicast islands via Mbone (Macedonia & Brutzman, 1994) and adding more control functionalities to the tunnel endpoints with DiveBone (Frécon, Greenhalgh, & Stenius, 1999), application level multicast (Hosseini & Georganas, 2004) has been proposed as an alternative solution without native IP multicast. Early adoption of IPv6 (Tapipamula, Grossetete, & Esaki, 2004) may make multicast become a reality in the internet.

The peer/server architecture is a hybrid of the two architectures above. There exists a server as in the client/server architecture. The server has the role of managing user membership and maintaining a consistent state of the virtual world. However, users directly communicate with each other using a multicast communication channel as in the peer/peer architecture. The server does not redistribute messages, but just receives them and updates the current state by joining corresponding multicast groups. This allows a newcomer to get the up-to-dated information of a virtual world from the server instead of multicasting the request for the information when he or she joins the world. This means that the peer/server architecture adopts the benefits of both the client/server model and the peer/peer model. SPLINE (Barrus et al., 1996) is an example of the peer/server model such that a locale server maintains information on a corresponding locale and objects in it, and notifies a new user of the latest update of the locale while users interact with each other via multicast assigned to the locale.

Our integration experiences of ATLAS with various applications, which will be described in detail in Section 3.4, show that no single communication architecture suffices for all DVE applications.

### 2.2 Interest Management

Though computing power and rendering speed are rapidly increasing, network resources still remain expensive compared with computational resources. As the number of users in a virtual environment increases, so does network traffic and thus latency as well. This would result in deterioration of the interactive performance of DVEs. Data losses due to the increase of network traffic can lead to inconsistency among users. To overcome these problems, various interest management schemes have been introduced (Morse, Bic, & Dillencourt, 2000). Some approaches (Oliveira, Crowcroft, & Diot, 2000; Levine, Crowcroft, Diot, Garcia-Luna-Aceves, & Kurose, 2000) consider the use of network level filtering, which are not the focus of our interest. Since they need not receive all the update messages related to a whole world, users instead want to receive only messages that they are interested in. Interest management is divided into a proximity based approach, a class based approach, and a hybrid approach (Greenhalgh & Benford, 1997; Singhal & Zyda, 1999).

Proximity based filtering mechanisms leverage human heuristics: humans do not recognize objects or interactions far from them. These mechanisms are mainly divided into region-based filtering and aura-based filtering. In the former (Barrus et al., 1996; Macedonia et al., 1995), a virtual world is divided into several disjoint regions. Each region is a multicast group and managed
by a manager called a region manager. A user then belongs to one region, and interacts with others in the region and/or in the neighboring regions. Since the user receives messages only from adjacent users in his region and/or those in the neighboring region, and not from all users in a whole virtual world, this method can reduce the number of users managed by the system and thus the number of messages exchanged in the system. However, this can perform only coarse-grained message filtering since the filtering is done with the unit of a region. In the aura-based filtering (Benford & Greenhalgh, 1997; Hagsand, 1996), each user in a virtual world has his or her own boundary of interest by the name of an aura represented by a range of vision or audition. The user interacts with only others whose auras collide with his or her own aura by forming a multicast group with them. Not only does this reduce the number of messages exchanged in the system, this method can perform fine-grained message filtering with a unit of a user. However, it requires relatively high local overhead due to continuous collision detection, and dynamic formation and deformation of multicast groups.

The class-based filtering scheme (IEEE Std 1516.1-2000, 2001) exploits application semantics. For instance, in military simulation, infantrymen are grouped into a platoon and they are not interested in communications among flight squadrons in the sky. The classes or interests of all objects that will participate in a virtual world are predefined. Objects in a virtual world are classified based on these classes. Users register their interests with corresponding classes before participating in the world and thus receive messages only from the objects of the registered classes. This allows more fine-grained message filtering than the proximity based mechanisms.

The hybrid approach (Abrams, Watsen, & Zyda, 1998; Oliveira & Georganas, 2003) combines the above filtering mechanisms for fine-grained filtering, and focuses on the balance between fine-grained data partitioning and computational overheads. Three-tiered architecture (Abrams et al.) provides three tiers of data filtering hierarchically. The first tier filters the data based on those regions that user is interested in. A region can be dynamically subdivided into eight smaller regions using octree when lots of objects are clumped in the region. The second tier uses protocol independent filtering. With generic information such as the area of interest (AOI) of a user, the second tier can reduce more data than the first tier. The third tier filters data with the most fine-grained manner using protocol specific information. The third tier allows users to receive only necessary data from others with any kind of a protocol that is designed to support specific filtering adapting to the characteristics of a target DVE system. While this approach reduces the number of messages, it requires a dynamic region split and merge scheme when many users are crowded in a region, which imposes heavy computational overheads even if the solution limits the number of partitions. VELVET (Oliveira & Georganas) aims to allow real time adaptation according to the local client’s needs using not only spatial distance, but also the local user’s interests. It supports multi-level AOIs with a metric chosen by each user. Since a user can choose the metric, a metric specifies a parallel virtual world (PVM) in which objects are placed according to the metric chosen by him or her. With the use of multi-level AOI and PVM, the system supports collaboration of heterogeneous users with different resource capabilities. Although the three-tiered architecture and the VELVET system use fine-grained filtering based on users’ interests, they assign as many multicast addresses as the number of users. This makes the system less scalable since multicast addresses are a limited resource.

As shown above, some systems such as collaborative engineering require no or coarse-grained interest management while other systems such as virtual community or large-scale simulation make dynamic use of coarse-grained and fine-grained filtering schemes. Therefore, the interest management should support various levels of filtering schemes.

### 2.3 Concurrency Control

Shared information in DVEs is often replicated at each user’s site to provide acceptable interactive performance, especially where users are geographically distributed over large networks such as the internet. Replication enables users to locally access and update the data.
On the other hand, the cost of replication is to maintain synchronization among replicas in the presence of multiple concurrent updates, which eventually lead to inconsistent views among users. As communication delay increases, the probability of conflicts between operations increases as well. With the assumption that users move around a virtual world at a normal walking speed, and that a user can manipulate an object only when near to it, if hundreds of users simultaneously request ownership of an object, it is hard for a system to resolve lots of concurrent requests in time. Therefore, concurrency control is required to maintain synchronization among replicas. There have been approaches that allow users to understand the limitation of latency (Conner & Holden, 1997; Fraser et al., 2000). However, they do not solve the problem systematically, but provide methods to let users know about current state and increase the time window to achieve the lock. Approaches to concurrency control have been broadly categorized into pessimistic, optimistic, and prediction schemes (Yang & Lee, 2000).

The pessimistic scheme (Hagsand, 1996; Barrus et al., 1996) blocks a user until a lock request for an object is granted and then allows the user to manipulate the object. It is simple and guarantees absolute consistency. However, when communication delay becomes high due to increase of the number of users, they suffer from the long response time, which in turn deteriorates the interactive performance.

The optimistic scheme (Sung, Yang, & Wohn, 1999) allows users to update objects without conflict checks with others and thus to interact with objects more naturally. However, when conflicts occur, a repair must be done. This not only makes systems complex but also perplexes users with undoing and redoing of previous actions on user interfaces.

The prediction based concurrency control scheme (Roberts & Sharkey, 1997) lies between the optimistic scheme and the pessimistic scheme. The keys to the prediction based concurrency control are twofold: to accurately predict a correct one among several users requesting an ownership and grant the ownership to it in time, and to pass the ownership to the next owner immediately when the current prediction turns out to be incorrect. However, in the existing scheme, the ownership requests are broadcasted and thus all the users in a world, including a current owner, receive the requests from all other users. As the DVE increases in size, so does the number of incoming messages at each potential owner and the network then becomes overwhelmed. Moreover, an owner must examine incoming messages that are not even destined for it. It may prevent the current owner from sending an ownership in time, and lead to wrong prediction.

Like other scalability aspects above, no specific concurrency control suits all applications. For example, collaborative engineering systems require a strict concurrency control because users must manipulate objects with a delicate and accurate manner. On the other hand, other applications such as virtual shopping mall demand prediction-based scheme if they focus on timely delivery of an ownership to the next owner among lots of candidates.

### 2.4 Data Replication

For supporting real time interaction in DVEs, it is common to replicate virtual world data from the server at the client. Surely offline distribution (via CD or other storage media) is prevalent in the online games, however, here we focus on online replication through which a client joining a virtual world downloads initial or updated world data from a server. Each client then updates its replicated data by local changes or notification of remote changes. As the size of the virtual world increases, significant transmission overhead increases the initial download delay of the virtual world data, especially when attempting to replicate a whole virtual world to the client. Some approaches use compression of the geometry data (Deering, 1995), or a combination of geometry and image data (Levoy, 1995) to save bandwidth. Several on-demand transmission (partial-replication) techniques have recently been devised (Capps, 2000; Chim et al., 1998) as another approach to reduce the overhead of full replication. In these techniques, instead of downloading the whole virtual world objects into the clients’ machines, only the objects a user needs are copied (Macedonia et al., 1997). A key
aspect in partial replication is how to efficiently replicate the required data lest that user’s immersion in the virtual world be disturbed for the loss of data. For efficient replication, two schemes are used together in general—prioritized transfer of objects, and a caching and prefetching technique (Park, Lee, Lim, & Yu, 2001).

The prioritized transfer of objects filters objects within a user’s viewing range and transmits only the objects that provide high fidelity to the user using level of details (LODs), or multi-resolution techniques. It maximizes the graphical fidelity of world objects as well as interactive performance by mediating the graphical detail and the transmission overhead of the objects.

Caching and prefetching make the demanded data immediately available in a timely manner by exploiting the locality of data even when the client’s access pattern changes over time. Since the performance of caching and prefetching depends on the priority of cached data, accurate prediction of a user’s behavior for the prioritization of world objects is an important factor to scalable data management in DVEs. Caching and prefetching approaches exploit the spatial relationship based on the distance between a user and objects. The spatial relationship is formed using the observation that the nearer the object lies to the user, the more probability it has to be accessed again. However, the approaches do not handle the scalability problem efficiently. The spatial relationship used in the existing approaches just guesses the user’s behavior from proximity between the user and objects. This makes it difficult to determine which types of objects are more important to an individual user, not reflecting the user’s interests—a significant factor affecting the user’s behavior. Since the diverse types of objects become extant as the number of objects in a virtual world increases, it is more difficult to correctly predict the user’s behavioral pattern according to the types of objects.

### 2.5 Load Distribution

Another approach for enhancing scalability is the multiple server architecture, adopted in several DVE systems as well as in many commercial multiplayer network games. Partitioning a virtual world into multiple regions and distributing the responsibilities for managing the regions across multiple servers can significantly reduce the workloads of individual servers, which enables a system to support more concurrent users and a larger virtual environment.

However, nonuniform distribution of users over the virtual environment would incur a workload imbalance among servers; that is, some servers handle more crowded regions than others and suffer from heavy workload. Consequently, users managed by heavily-loaded servers would experience low interactive performance due to long latencies of state updates at servers. To avoid such degradation of interactive performance, dynamic load distribution schemes have been introduced: overloaded servers transfer their excessive workloads to less-loaded ones to keep the interactive performance of their users acceptable. Current commercial online games are successful because they are particularly tuned for their specific games. However, too often the solution problems of current in the game industry is just to add another server machine when the current system cannot handle the load, even though there is a way to balance the load among the servers.\(^1\)

There have been many load distribution approaches in the distributed computing field; and only for multi-server DVE (MSDVE) systems, the existing dynamic load distribution schemes can be classified into three approaches: local, global, and adaptive approach. In the local approach (Ng et al., 2002; Pekkola et al., 2000), an overloaded server performs load distribution locally with only its neighboring servers, whose regions are adjacent to that of the overloaded server. The local approach incurs less overhead than the global approach because an overloaded server requires the information only on its neighboring servers and incurs a small amount of user migrations. However, in the case of a highly-skewed workload imbalance—that is, most overloaded servers are neighbored with each other rather than dispersed—an overloaded server cannot give its excessive workload to its neighboring servers because

---

1. This is based on personal communication with those who belong to NCsoft corporation and develop Lineage II (www.lineage2.com), one of the most popular MMORPGs all over the world.
they are also heavily-loaded. The local approach does not address such cases in an effective and timely manner. On the other hand, the global approach (Lui & Chan, 2002) can balance the workloads evenly even in a highly-skewed case because all the servers perform load distribution together with the global information. A central coordinator repartitions the whole virtual environment using a graph partitioning algorithm. However, the repartitioning overhead and the required amount of user migrations increase sharply as do the size of a virtual environment and the number of servers in the system. In the adaptive approach (Lee & Lee, 2003), an overloaded server balances its workload with a set of servers—beyond its neighboring servers—according to the workload status of servers.

Table 1 summarizes how the existing systems address the five scalability issues.

### Table 1. Summary of Existing Systems in Terms of Five Scalability Issues

<table>
<thead>
<tr>
<th>Systems</th>
<th>Communication architecture</th>
<th>Interest management</th>
<th>Concurrency control</th>
<th>Data replication</th>
<th>Load distribution</th>
<th>Target Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVE</td>
<td>Mixed model of client/server and peer/peer</td>
<td>Aura-based</td>
<td>Pessimistic</td>
<td>Partial replication</td>
<td>Not supported</td>
<td>Teleconferencing</td>
</tr>
<tr>
<td>MASSIVE</td>
<td>Unicast peer/peer</td>
<td>Aura-based</td>
<td>Not supported</td>
<td>Full replication</td>
<td>Not supported</td>
<td>Teleconferencing</td>
</tr>
<tr>
<td>MASSIVE-2</td>
<td>Multicast peer/peer</td>
<td>Aura-based, third party object</td>
<td>Not supported</td>
<td>Full replication</td>
<td>Not supported</td>
<td>Arena for online performance</td>
</tr>
<tr>
<td>MASSIVE-3</td>
<td>Client/server</td>
<td>Region-based (locale), and class-based</td>
<td>Pessimistic, optimistic, and prediction-based</td>
<td>Full replication</td>
<td>Not supported</td>
<td>Virtual sports arena application</td>
</tr>
<tr>
<td>NPSNET-IV</td>
<td>Multicast peer/peer</td>
<td>Region-based (hexagonal cell), and class-based</td>
<td>Not supported</td>
<td>Full replication</td>
<td>Not supported</td>
<td>Battle simulation</td>
</tr>
<tr>
<td>SPLINE</td>
<td>Peer/server</td>
<td>Region-based (locale)</td>
<td>Pessimistic</td>
<td>Full replication</td>
<td>Not supported</td>
<td>Virtual community</td>
</tr>
<tr>
<td>PaRADE</td>
<td>Mixed model of client/server and peer/peer</td>
<td>Not supported</td>
<td>Prediction-based</td>
<td>Full replication</td>
<td>Not supported</td>
<td>A ballgame</td>
</tr>
<tr>
<td>QUICK</td>
<td>Not supported</td>
<td>Aura-based and region-based</td>
<td>Not supported</td>
<td>Partial replication</td>
<td>Not supported</td>
<td>Virtual community</td>
</tr>
<tr>
<td>CyberWalk</td>
<td>Client/server</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Partial replication</td>
<td>Local distribution</td>
<td>Virtual community</td>
</tr>
<tr>
<td>OpenPING</td>
<td>Not supported</td>
<td>Supported, but not specified</td>
<td>Pessimistic and prediction-based</td>
<td>Full replication with varying replication rate</td>
<td>Not supported</td>
<td>Network game</td>
</tr>
<tr>
<td>Bamboo</td>
<td>Not specified</td>
<td>Not specified</td>
<td>Not specified</td>
<td>Not specified</td>
<td>Not specified</td>
<td>General purpose</td>
</tr>
<tr>
<td>ATLAS</td>
<td>Client/server and peer/server</td>
<td>Aura, region, and class-based</td>
<td>Pessimistic, optimistic, and prediction-based</td>
<td>Not specified</td>
<td>Partial replication</td>
<td>General purpose</td>
</tr>
</tbody>
</table>

3 **ATLAS**

In this section, we describe the scalability solutions that ATLAS provides regarding the issues discussed in Section 2 and the detailed architecture.

3.1 **Scalability Support**

3.1.1 **Communication Architecture.** ATLAS supports the peer/server model as primary communication architecture for versatility. A server joins several multicast addresses assigned to regions and maintains the membership of users and the state of the virtual world. A newcomer receives current membership and state information from the server through unicast. The newcomer can then send via multicast the update mes-
sages to other users in the region and, if required, the neighboring regions. ATLAS also supports the client/server model for web-based DVE applications where multicast is usually not assumed. The ATLAS server then performs filtering to compensate for the latency incurred by the role of reflector. In addition, except for the login process, ATLAS can support the peer/peer model. Joining a virtual world, a newcomer does not receive required information from a server. Instead, the newcomer requests it via a multicast address assigned to the current region. Then, a user who has the lowest ID (a representative user) responds to the request with his or her information. Interaction among users is done via multicast like the peer/server model.

As described in Section 2.1, it is up to application developers to select appropriate communication architecture, because one specific model cannot suit all DVE applications.

3.1.2 Interest Management. We assume that a whole virtual world is divided into several logical regions and that each user can communicate with other users in a region to which he or she belongs and those in neighboring regions. As described in Section 2.2, ATLAS supports multiple levels of interest management in terms of filtering granularity. For this, the ATLAS interest management scheme performs the filtering based on user interests and spatial distance (Han, Lim, & Lee, 2000). We leverage human heuristic such that, for instance, in a virtual shopping mall, users often tend to move to and crowd specific places with their own interests and to interact with those who have similar interests.

In our interest group based filtering scheme, we assume that messages generated by users are divided into two types: position-update messages that are generated when users move, and interaction messages that are required when users interact with each other. Each user has his or her interest area (IA) which is the same concept as DIVE’s aura (Hagsand, 1996). By using position update messages in a region and its neighboring regions, each user can detect IA collision, and her or she receives interaction messages incurred in the IA. We also assume that the oscillation problem can be solved by double-layered IA boundaries like the VELVET system (Oliveira & Georganas, 2003). When user comes into the IA of another user with the same interests, a group is created and they become members of the group. We call it a representative group area (RGA). An RGA helps to gather group information without any additional message transmissions for providing the group information to users that have low interest in the group. An RGA is always included in the IA of members of the group. Figure 1 shows the group creation of three users who have the same interests. Each RGA has a multicast addresses for high-fidelity interaction messages. Low-fidelity interaction messages are transmitted via an address assigned to a region. Each user in an RGA multicasts update messages to the rest of the group via a high-fidelity multicast address whenever interacting with the world. On the other hand, a representative user of the group who has the lowest ID among group members sends the aggregated update information of the group with low frequency to the user who has different interest. It enhances the interactive performance as the number of users in a DVE increases and/or users crowd in a specific place. Figure 2 shows an example of the scheme. A target user (represented as a dark circle) receives high fidelity data from the users of high interest (represented as a white circle). However, since rectangular and triangular users are not of high interest, their representative users (RU2 and RU3) send low fidelity data of their group to the target user.

We also support inter-region interactions with a more scalable manner (Lim & Lee, 1999). While a few systems (Barrus et al., 1996; Benford & Greenhalgh,
1997; Macedonia et al., 1995) support inter-region interactions, users have to pay the price: they must always be informed of the status of all the users in neighboring regions even if some of them are not of interest. This imposes communications overhead on the users who wish to pursue interactions with other users across regions and thus makes the system less scalable. In our inter-region interaction scheme, a region has a multicast address, and eight multicast addresses are also assigned to subregions of the region as shown in Figure 3. A user always belongs to three subregions: vertical (northern or southern), horizontal (eastern or western), and diagonal (northeastern, northwestern, southeastern, and southwestern) subregions. The region manager, which manages interaction among users in a region, then selects only a subset of users from the neighboring region whose members have a high possibility of interaction with users in the current region. This subset of users forms another multicast group. For example, when a user moves from a southern subregion to a northern subregion in a region, the related subregions are those in the northern and the southern neighboring regions. For the northern neighboring region, the user leaves the multicast address assigned to the southern subregion in the neighbor, and joins the one assigned to the neighboring region because the user approaches the neighboring region. This enables users in the region not to receive all the update messages from the neighboring region. They receive the update messages regarding only the users in whom they are interested in the neighboring region. For further improvement, the size of a subregion scope is dynamically changed according to the distribution of users in the neighboring regions. This helps the inter-region interaction not deteriorate the interactive performance regardless of not only the number of users but also their distribution as shown in Figure 4 (a). In addition, the subregions in the neighboring regions that are in the field of view of a user are only selected as shown in Figure 4 (b). This approach performs more fine-grained filtering using a fewer multicast groups with negligible computational overhead than the existing inter-region interaction approaches.

3.1.3 Concurrency Control. As described in Section 2.3, since one concurrency control mechanism does not suit all types of DVE applications, ATLAS supports both pessimistic (lock-based) and optimistic concurrency control. In addition, we devise a prediction-based scheme since the existing scheme (Roberts & Sharkey, 1997) does not scale in terms of delivering an ownership on time as the number of users increases.

Our entity-centric prediction based concurrency control scheme (Yang & Lee, 2000) satisfies the needs for scalability in terms of interactive performance as the number of users increases. We assume that a user can interact only with objects nearby. As shown in Figure 5, only the users surrounding a target entity multicast the ownership requests by using the multicast group address assigned to the entity. The number of messages per owner decreases drastically since the owner receives only from users joining the entity multicast group instead of from all users in the same region. This reduces network bandwidth consumption. Therefore, an owner makes a prediction with the reduced number of messages, which, in turn, results in a short request processing time. It allows the owner to determine the next owner and pass the ownership in time. In Figure 5, user A is the current owner of an object. After finishing the job, user A predicts a next owner based on the information in the candidate queue. There are four candidates in the queue: user B, user C, user D, and user E. User A grants the ownership to user C who has the second lowest predicted collision time and a positive direction to the target object. Even though user B has the lowest predicted collision time, that user’s direction is negative.
The performance of the scheme is improved further by grouping closely gathered entities into one entity group and sharing a multicast address among group member entities (Lee, Yang, & Hyun, 2000). It reduces the number of frequent join and leave operations, and maintains enough interactive performance. Another enhancement is made to support users with various navigation speeds (Lee, Lee, Han, & Hyun, 2001). A separate entity radius is assigned for a different speed and a separate queue is allocated for the users of each speed. Each queue is examined in parallel to predict the next owner candidate and among the selected candidates is chosen the final candidate, which contributes to the timely advanced transfer of ownership by using an appropriate entity radius based on a user’s speed.

### 3.1.4 Data Replication

Several data replication schemes (Capps, 2000; Chim et al., 1998) using partial replication do not handle the scalability problem efficiently. The spatial relationship used in the existing approaches just guesses the user’s behavior from proximity between the user and objects. This makes it difficult to determine which types of objects are more important to an individual user, not reflecting the user’s interests—a significant factor affecting the user’s behavior. Since the diverse types of objects become extant as the number of objects in a virtual world increases, it is more difficult to correctly predict the user’s behavioral pattern according to the types of objects.
For efficient data management of a virtual world, we have proposed a scheme using user-based caching and prefetching exploiting the object’s access priority generated from spatial distance and individual user’s interest in objects in DVEs (Park et al., 2001). The scheme leverages the locality obtained from interactions between a user and objects during the user’s navigation in a virtual world, so called “user-based” data management. We assume that the behavioral pattern of a user is explained using the user’s interest in objects in the world. To incorporate the user’s interest into the access priority of objects, we leverage the fact that a user tends to repeatedly visit objects interesting to that user or highly popular objects likely to attract the user. To enumerate the level of interest and popularity of an object, we introduce two values, interest score and popularity score of an object. The interest score of an object is set per user and represents how much the user expresses interest in the object. The popularity score of an object is set per world and represents how many people in the world express their interest in the object. By combining these two values with the spatial relationship, we improve the performance of caching and prefetching since the interaction locality between the user and objects are reflected in addition to spatial locality. Interest and popularity scores are determined by the number of access times for a given object. Whenever a user accesses an object, the client notifies the server of the interaction. The server then updates the access count of the object for the user. Until the interest and the popularity scores are sufficiently cumulated, the access priority is determined only by spatial relationship between users and objects. For further improvement of the cache hit rate, we incorporate the user’s navigation behavior into the spatial relationship between a user and the objects in the cache. We observe that a user usually alternates a navigation mode between wandering and moving. In the moving mode, a user continuously moves around a virtual world. On the other hand, the wandering mode means that a user stops moving and looks around at the surroundings. An object residing at the user’s moving direction should have the same priority as an object residing at the opposite side of the user’s moving direction in case of a wandering mode since it implies the possibility of rapid rotation. Apparently the former object should have higher priority than the latter object in case of a moving mode. This provides more accurate information for caching and prefetching.

3.1.5 Load Distribution. ATLAS supports a scalable dynamic load distribution scheme for MSDVE systems, where users are highly skewed rather than uniformly distributed over a virtual environment (Lee & Lee, 2003). Unlike the existing local approach, an overloaded server in our scheme performs load distribution with a set of servers—beyond its neighboring servers—which are chosen dynamically, adapting to the workload status of servers. Also, our scheme is different from the global approach in that it selects only a subset of servers, just enough to resolve the excessive workload of the overloaded server. The server initiating load distribution first selects a set of servers to be involved in load distribution. It chooses a least-loaded one among its neighboring servers and requests the chosen one to participate in load distribution. The chosen server rejects the request if it already participates in or performs another load distribution; otherwise, it participates in load distribution by sending the workload information on its neighboring servers to the initiating server. If the nominated neighboring server is not capable of taking on the excessive workload of the initiating server, the selection is performed again among the neighboring servers of not only the initiating server but also the chosen neighboring server. The selection continues until the excessive workload can be resolved. Then, the regions dedicated to the involved servers are repartitioned using a graph partitioning algorithm so that all the servers involved in load distribution will have the same number of users, that is, the workloads are evenly distributed. The involved servers then migrate their users with each other in a peer/peer manner according to the result of repartitioning.

3.2 Implementation

ATLAS implementation consists of four modules that support the scalability schemes described above as illustrated in Figure 6: communication manager, event
manager, session manager, and region manager. The communication manager is used for communication channel management. Depending on the communication type required by a target DVE application, ATLAS can be organized as a client/server based or a peer/server based system. The region manager consists of submodules that support interest management, consistency management, data distribution management, and load distribution management. All the actions in ATLAS are defined as an event. The event manager mediates communications between peers or peers and a server and delivers an event to a proper manager module(s). The session manager manages membership in a session and the region managers in it.

ATLAS is implemented in C++ and Java on Linux, Windows, and Irix (http://cds.icu.ac.kr/atlas/resources.html). Since it has little dependency on a specific platform, the C++ version of ATLAS can be easily ported to other platforms such as various UNIX versions with a little modification. Figure 7 shows a set of classes, and the detailed implementation of the classes is described in Lim, Han, and Lee (2001).

3.2.1 Initialization of ATLAS. ATLAS is initialized with a configuration file containing information such as system type—whether the application is server or client—and the IP address and the port number of the server that clients have to initially connect to. The configuration file of the ATLAS server has more information such as the communication architecture, structure of sessions and regions in each session and so on. ATLAS initializes the communication manager, the event manager, and the session manager in turn. If more than one session are required to exist at the same time, a separate session manager is initialized for each session. If the system type is a server, the session man-
ager initializes its corresponding region managers. If the communication architecture is the peer/server model, the region managers join the multicast address assigned to each region because the server should receive all the update information from all clients to keep a consistent state among them. If the system type is client, the session managers do not initialize the region managers. A region manager and its neighbors are initialized; after that the client selects a session, and receives region information from the server. If the communication architecture is the peer/server model, the client also joins the multicast addresses assigned to its region and neighboring regions.

3.2.2 ATLAS Events. Peers or servers communicate with each other via ATLAS events. There are six types of ATLAS events: session event, region event, data event, interest event, concurrency event, and dummy event. Each event type has its own purpose. For instance, a session event is defined for session-related events such as user login and logout. The type of an event also implies which module will handle the event. For instance, a session event is processed by the session manager. Each event type has several different event IDs according to the purpose of an event. For instance, an interest event has more than ten event IDs such as USER MOVED, USER LEAVED, and so on. Each event class provides interfaces for marshalling or unmarshalling a message. The event manager dispatches events to a proper handler. The detailed information on each event type is described in Lim et al. (2001).

3.2.3 Communication Manager. The communication manager is responsible for creation and deletion of communication channels and message transmission. For channel management, the manager owns a channel list which holds the channels currently being open. It supports polling and threaded socket groups to receive incoming messages. To send outgoing messages, the communication manager provides various communication methods such as unicast, multicast, and broadcast. Since we abstract a raw socket interface to facilitate easy communication channel management, it is possible to extend ATLAS basic socket classes for providing various communication protocols such as reliable multicast and stream service. In addition, the communication manager handles communication exceptions to provide robust and stable communication channels.

The ATLAS clients interact with each other by sending messages according to the selected communication architecture. While multicast is used in the peer/server model, unicast via the server is used in the client/server model. Receiving a message from a client, the ATLAS server determines whether it forwards the message to other clients according to the event type and ID.

3.2.4 Event Manager. The event manager mediates events between the communication manager and high-level components, such as the session manager, the region manager, the interest manager, the concurrency manager, the data distribution manager, and the load distribution manager. It extracts the event information from a message delivered by the communication manager. It then checks the type and event ID field of the event to find an appropriate handler. The event is deliv-
cred to the chosen handler which is registered to the event manager. The structure of the event manager is illustrated in Figure 8.

All ATLAS managers inherit the CAtlasHandler class to receive and process events from the event manager or other components. A manager that wishes to receive a specific type of events must register its reference to the components from which the manager is to receive events. The event manager has another role of passing outgoing events to other remote peers or servers. High-level managers such as session managers and region managers send their events via the event manager. Then it selects channels corresponding to the destinations of the events and passes them to the communication manager.

3.2.5 Session Manager. A session in ATLAS implies a basic unit which is an independent virtual world. The session manager provides users with the interfaces for entering or leaving its virtual world and membership management, and defines specific rules applied to a session. Since we assume that a virtual world is divided into several logical regions, the session manager holds references to the region managers that manage logical regions. It inherits the CAtlasHandler class in order to handle incoming events and to send session specific events to session participants or another server.

ATLAS also supports multiple sessions. For dynamic management of sessions, the multi-session manager holding a reference list of session managers provides users with the interfaces for initiation, termination, selection, join, leave, creation, and deletion of a session. To enter a virtual world, users query the multi-session manager about the information on available sessions, and select a session among the sessions maintained by the ATLAS server. Receiving a login request from a user, the session manager verifies the user with his/her name and password. It then informs the user of regions in it, among which the user enters a default region. After receiving information on other users and objects in the region, the user can start to interact with them.

3.2.6 Region Manager. The region manager plays a major role of keeping a consistent state or view among users who participate in a virtual world. For this, it keeps track of all state information including dynamic objects and users in a region and its neighboring regions. Figure 9 illustrates a basic structure and event flows in the region manager.

The region manager has four modules each of which contributes to the scalability described in Section 3.1: interest manager, concurrency manager, data manager, and load distribution manager. The detailed internal structure of each module is described in the following subsections.

3.2.7 Interest Management. In this Section, we present the implementation details of the proposed schemes in the interest manager component of ATLAS.
The related classes are CAtlasIRIManager, CAtlasIManager, and CAtlasUIGBManager.

### 3.2.7.1 Subregion Management

The CAtlasIRIManager class manages the inter-region interaction management scheme of ATLAS. The main task of the CAtlasIRIManager is to make each user join and leave appropriate multicast addresses assigned to subregions according to which subregions the user belongs to.

When a user joins a session or moves to another region, the inter-region interaction scheme is initialized. A user whose reference is transferred as a parameter joins multicast addresses of a subregion which is adjacent to the user if the user is located far from the neighboring region which contains the subregion, so that the user may receive only update messages of other users which are adjacent to the current user. If the location of the user is adjacent to a neighboring region, the user joins the multicast address which is assigned to the neighboring region because the user in this case is interested in the update messages of all users in the neighboring region.

Figure 10 illustrates an overall execution flow of the inter-region interaction scheme. In Figure 10, whenever a user moves and sends a position update message in a region, the user checks whether to change interest or not according to the current position, Figure 10 (1). If the new position of the user is still in the current region, the user then checks whether to change the current subregions, Figure 10 (2). If the user changes any subregion, then the user leaves the previous subregions and joins the new subregions, Figure 10 (3). If the new position is out of bounds of the current region, the user changes the current region and subregions, Figure 10 (4,5).

### 3.2.7.2 Aura Management

The CAtlasIAManager class manages collision of interest areas (IAs) among users in a region when ATLAS uses a user interest group based filtering scheme. Figure 11 shows the overall execution flow of the interest group based filtering scheme. This scheme performs IA collision detection among users.

If another user is included in the IA of the current user, then the CAtlasIAManager class checks if the user has the same interest as the current user, and checks if the user is in interestUserList or uninterestUserList, Figure 11 (1). If the user has the same interest as the current user and is not contained in any list, then the user is classified as a new user from whom the current user can receive high fidelity data. So it is added to interestUserList, Figure 11 (2). But if the newcomer does not share the same interest with the current user, it is added to uninterestUserList, Figure 11 (3).

If a user leaves the IA of the current user, the CAtlasIAManager class checks if the user was in its interestUserList, and uninterestUserList, Figure 11 (4). If so, the user is removed from interestUserList or uninterestUserList, Figure 11 (5,6).

### 3.2.7.3 Interest Group-based Filtering Management

When two users come into the IA of each other with the same interest in the case of the user interest
group based filtering scheme, a group is created and the CAtlasUIGBManager class manages the group. As shown in Figure 11, when a user comes in IA of the current user, the CAtlasUIGBManager class checks the state of a newcomer based on the position of the current user, Figure 11 (7). If the newcomer is not a member or a representative user (RU) of a group, the user who has higher priority creates a group and the other joins the group. The priority of the user is determined based on the duration in the group and object ID.

Whenever a user enters a virtual world, the session manager notifies that user of an increased unique ID. Users in a region multicast their identity data (e.g., object ID) to all the other users via the multicast address of the region whenever their positions get updated. This allows each user to be aware of who has higher priority. If the newcomer is already a member or a RU of a group, it means that the current user has lower priority with the same interest, so the user becomes a member, Figure 11 (11).

If the current user is not a member of a group, which indicates that the state of the user is independent, and the other user has lower priority than the user, the CAtlasUIGBManager class sets the state of the user to requesting and sends a message to the region manager for checking conflict, Figure 11 (8). If the region manager confirms no conflict, the state of the user is set to RU. Then the user sends a message to notify that a new group is created and the user becomes an RU, Figure 11 (9). If the current user has lower priority than that of the newcomer, then the current user becomes a member of a new group, Figure 11 (10).

A RU manages the membership of a group. If a user has the same interest with the RU and it is out of the representative group area, the user is not a member of the group anymore. Thus, the RU of the group removes the user from group list and the RU sends a notification message to the user, Figure 11 (12). The user who receives the message sets the RepUser field as itself and sets the user’s state to independent. The membership management is processed in RUs. It reduces computational overhead because other members of the group do not need to perform group management.

When the states of a user such as a position are changed, the user sends update messages. If the user is an RU, the user sends messages using the region multicast address and the RGA address; otherwise, the user sends messages only to the region multicast address.
3.2.8 Concurrency Control. The concurrency manager implemented as CATlasConcurrencyManager class is initiated by the region manager at login time. Figure 12 shows the execution flow of the concurrency manager of a user and that of an owner. Whenever a user moves in a region, the concurrency manager updates the user’s current position and maintains the previous position for the purpose of calculating the moving direction of the user Figure 12 (1). The concurrency manager then investigates whether the user collided with the surrounding of an entity according to the user’s position and speed, Figure 12 (2).

When the concurrency manager detects collision, it multicasts a join message as an implicit ownership request, Figure 12 (3). When a current owner receives the join message, it adds the user to a candidate queue which is separated according to a user’s speed. If a user is already a candidate of an entity, the concurrency manager checks whether the user collided with the entity and departs from the surrounding of the entity, Figure 12 (4). When the candidate departs from the surrounding, it leaves the multicast group assigned to the entity and does not receive concurrency control events related to the entity anymore, Figure 12 (5). When a candidate collides with an entity without an ownership, it informs the current owner of the collision through the multicast, Figure 12 (6). If the owner does not interact with the entity, the owner immediately relinquishes the ownership to the candidate, Figure 12 (7).

A current owner determines whether it finishes interaction with an entity by getting a user’s moving direction based on the previous and the current distance to the entity, Figure 12 (8). If the owner moves in a NEGATIVE direction, which means that the owner has finished the interaction, it predicts a next owner, Figure 12 (9).

If the navigation speed of a user can vary, the current owner maintains as many candidate queues as the number of different speeds, and prediction is achieved based on each candidate queue according to navigation speed in parallel. The concurrency manager determines how far a candidate is from the entity, whether it moves in the direction of the entity, and how much it is interested in the entity. The concurrency manager then gets a reference of a candidate who is the predicted next owner with minimum predicted collision time, POSITIVE direction, and STRONG interest.
### 3.2.9 Data Replication

This section describes details of the data manager module of ATLAS, the user-based caching and prefetching scheme. Figure 13 illustrates overall execution flow of the data manager. Since the data manager module is asynchronous, it does not have an impact on the application in terms of rendering.

The data manager is implemented in CAtlasData-Manager class which contains CAtlasCache class, and uses it to provide partial replication and cache management. Whenever a user moves on to the next position the application requests the data manager to check objects in the viewing range of the user, Figure 13 (1). The data manager scans the user’s cache to determine whether all of the objects are in the cache or not, Figure 13 (2). If all of the objects exist, the data manager notifies the application to make ready for rendering, Figure 13 (3). Otherwise, it sends an event to server for downloading missing objects, Figure 13 (4). During the process, the data manager updates the access count of the world database and the user database stored in the cache. The information of access count is sent to the server for maintaining the interest and popularity of objects after the session is ended up.

The CAtlasCache class contains three member variables of object table, worldDB, userDB, and cacheTable. The first two variables provide popularity and interest of objects, respectively. The cacheTable represents the cache structure. Whenever a user moves, the data manager finds objects for rendering, and provides the cache with the list of objects for updating access count of objects. The CAtlasCache class maintains the information for prioritizing objects in the cache, and uses the information for efficient cache management. The CAtlasCache class checks whether the capacity of the cache is sufficient to contain objects, Figure 13 (5). If the cache is enough, in other words, if the cache is not full, the class inserts the objects into the cache, Figure 13 (6), and updates the access counts of the objects for the calculation of the access priority, Figure 13 (7). If the cache is full, cache replacement is performed. The replacement algorithm finds the lowest priority of object in the cache, Figure 13 (8), and deletes it, Figure 13 (9). Until the capacity of the cache is enough (as long as it is not full), the process is repeated.

### 3.2.10 Load Distribution

The implemented load distribution scheme of ATLAS requires a session server and region servers connected by a high-speed network. For a MSDVE system, we separate the session management and the region management from a server. The session server has a role of the former, and the region server of the other. In addition to the fundamental
functionalities of the session management, the session server manages information on cells, that is, which area of a VE a cell occupies and which region a cell belongs to. The main part of the load distribution, a load distribution manager implemented in CAtlasLoadDistManager class, is managed by a region server which keeps a consistent state in its dedicated region using various supporting modules such as the interest manager, the concurrency manager, and the data manager.

After the session server is initiated, it retrieves the session information which describes how a VE is subdivided by cells. It then initializes the cells and regions. The region server connects to and registers its information in the session server. The session server allocates a set of cells (a region) and sends their information to the registered region server. The region server then initializes the region from the received information. The load distribution manager in each region server performs dynamic load distribution with others, as shown in Figure 14. Periodically, a load distribution manager evaluates the workload of its region server, Figure 14 (1) and exchanges such information with others in the neighboring servers, Figure 14 (2). When receiving workload information from a neighboring server, a load distribution manager updates an instance of the CAtlasLoadInfo class which contains the workload information of a server. If a server gets overloaded, it initiates load distribution. It first selects a server from its neighboring servers with least workload and sends a request to the chosen neighboring server, Figure 14 (3). The chosen server replies to the requesting server with the message that it can participate or not in load distribution, Figure 14 (4). After completing the server selection, the initiating server executes a graph partitioning algorithm to repartition the regions, Figure 14 (5). The repartitioning result is disseminated to the involved servers, Figure 14 (6). The involved servers then perform cell migrations, Figure 14 (7). A region server sends the state information of users in the cells that should be migrated to another region server. Finally, the load distribution managers of the involved region servers notify their users inhabiting the migrated cells of the information on the changed region server. The load distribution manager of a client handles the notification from the region servers. They also send the updated information on their regions to the session server, Figure 14 (8).

3.3 Performance Evaluation

In this section, we describe how performance evaluations were performed in terms of simulation models and performance metrics, and the simulation results of our schemes in four scalability problems: interest management, concurrency control, data replication, and load distribution.

3.3.1 Simulation Model. Our scalability solutions were simulated with the following simulation model. We constructed a simplified virtual environment with regions. A region is a rectangle whose size is $250 \times 250$ pixels. We ran the experiment on two Windows 2000 machines connected by 100 Mb Ethernet. One machine ran an ATLAS server, and the other ran an ATLAS client. The simulation program emulated multiple users generated as non-playable characters by the server. The users were randomly created and moved in random directions. We assumed that the speed of all users was fixed with 5 pixels per 200 ms. A thousand objects per region were uniformly distributed. We as-
sumed that there is a dynamic allocation method of available multicast addresses, and statically assigned them in the evaluation. The join/leave cost is the same issue as other existing approaches adopting multicast. The majority of the join/leave cost results from the join latency. The leave process does not incur any latency, just bandwidth wastage until pruning. The join latency can be reduced by adopting a small Unsolicited Report Interval of IGMP v3.

3.3.2 Interest Management. For an evaluation of the proposed interest management scheme, we constructed a world with nine regions. When the interest group based filtering scheme was executed, users’ interest values were randomly created, and a low frequency message rate made up 30% of the high fidelity message rate. The radius of interest area (IA) of a user was 100 pixels. The region-based and aura-based filtering schemes were comparable with the proposed schemes in terms of the number of transmitted messages and computational overhead. The hybrid approaches were not included because they proposed a policy, not a specific mechanism for fine-grained filtering. Instead, we analyzed the number of multicast addresses and join/leave operations for the hybrid approaches.

Figure 15 shows the average number of messages that a user receives per second. Our interest group based filtering and inter-region interaction mechanism (uig + inter-region management) receives fewer messages than the existing approach (aura-based + no inter-region management). The proposed schemes can also be optionally used to support various filtering levels from coarse-grained to fine-grained like a multi-level filtering scheme. If we use only the inter-region interaction scheme without adaptable scoping (inter-region management with no adaptable scoping), messages are filtered only by subregions in neighboring regions and the filtering level is low. As the adaptable scoping approach is added (inter-region management) and both the inter-region and intra-region interaction schemes are combined (uig + inter-region management), the filtering level becomes high.

The user interest group based filtering scheme affects the number of received messages according to the rate of a low frequency message as shown in Figure 16. A user can select various rates according to the desired intention. If the user wants to see the detailed behavior of other users who have different interests, a user sets the high rate of a low frequency message. For example, when the rate is 100% of the high fidelity message rate, it means that a user wants to see all the interaction of others in the interest area with high fidelity, but the user receives all the messages in which the user is not interested. If the user does not want to see the detailed behavior of others with different interests, the user sets the low rate. When the rate is 0%, it means that a user wants to interact only with others who have the same interest with him. However, a user does not know where other users with different interests are. If the user wants to see
not the detailed interaction of others with different interests but the rough behaviors, a user selects a low frequency rate between the above two extremes. We set the low frequency rate to 30% of the high frequency rate, since it is low enough for a user to recognize in which direction other users move. If the rate is lower than 30%, other users seem to appear and disappear arbitrarily.

While the proposed schemes reduce the number of messages, they may appear to incur more local processing overhead than the existing methods. The overhead results from the subregion check, switching from one multicast address to another for inter-region interaction, and from grouping in the interest group based filtering scheme. Figure 17 shows the differences of the local processing time of an update message on average. The range of computational overhead is less than a few milliseconds regardless of the number of users. Although the proposed scheme dynamically manages subregion boundaries, resizing subregions in the user’s region and joining subregions in neighboring regions in the user’s viewing direction are local operations of each user. There is no need to reconfigure regions, which requires changing the total number of regions and reforming multicast groups, and has to be notified to all the affected users. Therefore, there is no significant difference of the computational overhead between our approach (uig + inter-region management) and the existing approach (aura-based + no inter-region management).

For bandwidth wastage, even though our schemes have more superfluous data than the VELVET system, it is due to position-update messages. If users are not interested in what other users do, but are interested in where they are in their interested area, it is not bandwidth wastage. Another evaluation metric for interest management is the usage of the multicast addresses. For simplicity, we assume that a region is a rectangle with $100 \times 100$ pixels, that a user’s interest area (IA) radius is 10 pixels, that a hundred users are located in the region, and that all users have the same interest type.

In the region-based filtering scheme, one multicast address is assigned to a region since the region forms one multicast group. Instead of a small number of addresses, a user joining a multicast group must receive unnecessary messages, even if the user is not interested in them. The hybrid approach is another extreme case such that it must assign 100 multicast addresses in a region because each user has its own address for specific filtering while filtering messages in a most fine-grained manner. The aura-based filtering scheme is somewhere between the above two extremes. The number of multicast addresses is determined by the size of the interest area (IA). If a radius is 0, there is no one who is interacting with the others, and each user has its own multicast address. If the IA covers the whole region, there is only one multicast group because all the users in the region are located in the IAs of others. According to the above assumptions (a region is a rectangle with $100 \times 100$ pixels, and a user’s IA radius is 10 pixels), the radius of the IA is one tenth of a segment of the region. Twenty-five multicast groups are then formed in a region in the worst case, as shown in Figure 18.

Our schemes belong to both the region-based and aura-based methods in terms of address usage. The proposed inter-region interaction scheme assigns multicast addresses per region and subregion. Because a region has eight subregions, our inter-region interaction scheme requires nine addresses per region. The proposed intra-region interaction scheme forms an interest group only when users are located in each other’s interest area with the same interest. If all users have different interests, there is no additional required address since no interest group is formed. In the worst case in Figure 18, if there are 50 interests and each 2 of 100 users have
the same interest, our scheme forms 50 interest groups and requires the same number of additional addresses. It means that the required number of addresses in the worst case is half of the number of users. Although it assigns more addresses (59) than the aura-based approach (25), the proposed schemes use many fewer addresses than the hybrid approach (100).

The allocation of additional multicast addresses may cost a lot though it costs less than allocating the address per object. However, we expect that multicast addresses will not be expensive resources in the internet over which IP version 6 (Tapiamula et al., 2004) is expected to be deployed widely.

3.3.3 Concurrency Control. We compared our entity-centric prediction based concurrency control with the existing prediction based scheme, PaRADE (Roberts & Sharkey, 1997) in a region. Important metrics were the number of transmitted messages for ownership requests and grants, and correct prediction probability. We set the size of an entity radius as $2 \cdot (\text{maximum latency} \cdot \text{user speed})$ which maximizes the probability of correct prediction as we evaluated the probability by varying the size of the entity radius. Time among systems was synchronized by NTP.

Figure 19 shows the number of messages exchanged for ownership requests and grants for 10 s. It clearly shows that the entity-centric approach gives better scalability as the number of users increases.

Figure 20 shows that the correct prediction probability of PaRADE is a little higher than the one of ATLAS due to recovery probability. In the proposed scheme, all users send a join message as an implicit ownership request at entity radius, regardless of their latency. Latency is a delay required for exchanging a concurrency control message between an owner and an owner candidate, and includes message generation and reading time as well as communication delay. However, in PaRADE, each user calculates the request send time based on latency. Therefore, the request send time in PaRADE is closer to the time when the ownership is needed than is the case for ATLAS. The recovery cost of PaRADE is lower than that of ATLAS. This alone is why the correct prediction probability of PaRADE is slightly higher (by a few percent) than that of ATLAS.

Figure 21 compares interactive performance between ATLAS and PaRADE. “On time ownership passing” means that a user has received an ownership
just before it reaches the target entity, and thus it can manipulate it immediately. PaRADE fails to provide enough interactive performance to the number of users because the current owner cannot send ownership to the next owner in time due to large network traffic and message processing overhead. On the other hand, ATLAS provides enough response time even as the number of users increases.

3.3.4 Data Replication. The proposed data replication mechanism was compared with the existing spatial based cache replacement scheme, MRM (Chim et al., 1998). The performance of the data replication was measured by cache hit rate—how well the replacement algorithm retained the objects that a user would see in the near future. Another metric was cache hit rate for interested objects—how many times a user immediately saw the objects interesting to him. We set a virtual world with 16 regions, and the viewing angle of a user to 120°. Cache size was set to basic cache size, which is calculated as $n \times (M'/M)$, where $n$ is the number of objects in a virtual world, $M$ is the size of the virtual world, and $M'$ is the size of portion of the virtual world within a user’s viewing range at one time (Lee, Kim, & Park, 2001).

Figure 22 and Figure 23 show the comparison of hit rates between the proposed scheme and the existing scheme, MRM. We simulated user navigation 200 times, and averaged the hit ratios per 10 times. The figures show that the proposed data management scheme improves both the total hit ratio and hit ratio for interested objects by about 10% on average when the cache size is set to the basic cache size. Since it is important that a user immediately see the objects interesting to that user, the hit ratio for interested objects is a true measure of data management in DVEs.

3.3.5 Load Distribution. For the load distribution scheme, we assumed that a server manages one region, the capacity of a server (CP) is 250 users, and a region consists of $5 \times 5$ rectangular cells. We evaluated distribution performance with increasing the number of servers (Ns) from $4 \times 4$ (16) to $12 \times 12$ (144). The
total number of users was set to $0.8 \times CP \times Ns$. This means that the average workload of a server is 80% of the server capacity. Users are assumed to be uniformly distributed and to randomly move around a virtual world. Since we achieved high distribution performance with less overhead when we set the distribution threshold to the same as the workload of a server, the distribution threshold was also set to 80% of the server capacity. We compared the proposed load distribution scheme with the existing local scheme in which only neighboring servers are utilized in the distribution, and the global scheme in which all servers are utilized. We took effectiveness and overhead as performance metrics. Effectiveness is enumerated by how much a scheme reduces the overloaded users, while overhead is indicated by how much a dynamic load distribution scheme incurs burdens to the servers.

Effectiveness is defined as:

$$\text{Effectiveness} = \frac{(R_{OU} - R_{OU}^*)}{R_{OU}^*} \times 100\%$$

where $R_{OU}$ is the ratio of overloaded users (the number of overloaded users divided by the total number of users), $R_{OU}$ is the average $R_{OU}$ during simulation, and $R_{OU}^*$ is the average $R_{OU}$ when the servers do not perform dynamic load distribution. Overloaded users are those who are managed by overloaded servers. The larger $R_{OU}$, the more users suffer from long update latencies imposed by the overloaded servers.

Overhead is defined as:

$$\text{Overhead} = \frac{R_{MU}}{100\%}$$

where $R_{MU}$ is the ratio of migrated users (the number of migrated users divided by the total number of users), $R_{MU}$ is the average $R_{MU}$ during the simulation. Migrated users are those who migrate from an overloaded server to an available server as a result of dynamic load distribution. The larger the $R_{MU}$, the more users should migrate and the more burdens are imposed to the participated servers.

Figure 24 and Figure 25 show the effectiveness and overhead of the proposed scheme as well as existing local and global schemes, as the number of servers increases. As shown in Figure 24, the global scheme ensures very high effectiveness (close to 100%)—that is, all the overloaded users are resolved, regardless of the number of servers. Although the proposed scheme exhibits about 10% lower effectiveness than that of global scheme, it still guarantees high effectiveness (more than 90%) for all the cases of $Ns$.

Figure 25 compares the overhead of the proposed scheme and the existing local and global ones. For all the schemes, the overhead increases as the number of servers does. But the overhead of the global scheme
increases much more rapidly than that of the proposed
and local ones as does Ns.

In summary, the proposed scheme exhibits relatively
high effectiveness with low overhead regardless of the
size of the system.

3.4 Integration of ATLAS with
Applications

Various DVE applications can be built based on
the ATLAS classes described in the previous section.
Depending on whether an application has a role of cli-
ent or server, it uses the APIs of ATLAS Client Stub or
ATLAS Server Stub, respectively. A server application
integrated with ATLAS can provide different user inter-
face and application specific information to users ac-
cording to the requirements of application developers.
A client application developer can also select any appli-
cation type, rendering subsystem, and user interface.
First, the application developer should add the ATLAS
library to the application, assign a stub instance, and
register a function that processes the receiving events
forwarded by ATLAS. After that, the developer can start
ATLAS. In the case of a client application, the further
steps are required; logging on to a server, requesting
available session information, and joining a session and a
region. While interacting with other clients, the applica-
tion can send events via the stub module and can pro-
cess receiving events by the registered functions. When
the registered receiving function in the application re-
cieves an event, it is totally up to the application devel-
opers how the application must handle the event; it can
update application specific data, or call a rendering sub-
system to present the effect of the event to users. The
client application can also leave the session, log out
from the server, or disconnect itself from the server.

Figure 26 illustrates overall integration sequences for
integrating a client application with ATLAS. ATLAS
provides application developers to choose an approach
per each of five scalability components using a configu-
ration file. In addition, current implementation allows
users to change interest management scheme during
runtime by providing APIs in the ATLAS stub module.
For example, a server application whose clients receive

| Implement a receiving function |
| Register function |
| Assign stub |
| Start ATLAS |
| Login |
| Create ATLAS event |
| Send |
| Terminate ATLAS |
| Connect to server |
| Process events |
| Logout |
| Disconnect from server |
| call registered function |

Figure 26. Integration sequences of client applications with ATLAS.

update messages from their region and the neighboring
regions can change its current interest management pol-
icy to the subregion based inter-region interaction
scheme when the number of users becomes large.

In the following subsections, we describe our experi-
ences that we have obtained from the integration of
ATLAS with various applications.

3.4.1 ATLAS + Collaborative Engineering
System. Kitten (http://vr.kaist.ac.kr/Kitten/), devel-
oped by VR lab at KAIST, is a graphic library to imple-
ment a single user virtual reality system. It consists of a
rendering module, a simulator module, and a user inter-
face module, each of which runs as an independent
thread. We integrated ATLAS with Kitten for a collab-
orative engineering application which took about two
months as the first integration experience. Integration is done with help of the intermediate veneer layer composed of ATLAS-Kitten Integrator and ATLAS Stub.

A client application is designed to enable users to visualize large CAD data and manipulate it for discussion on design decisions. It is built using the peer/server model and implemented in C++. A server performs user membership management and conflict resolution. For conflict resolution, a simple lock based concurrency control scheme is adopted since it is suitable for detailed and correct manipulation of CAD visualization data. Since a collaborative engineering system requires only a small number of users, no interest management scheme of ATLAS is applied to the application. We assume that CAD visualization data is fully replicated in each client. Figure 27 is an image capture of the application.

3.4.2 ATLAS + Virtual Community. Virtual Playground (VP; Schwartz et al., 1998), developed by the Human Interface Technology (HIT) laboratory at the University of Washington, provides a shared virtual world that demonstrates how people learn, perform cooperative work, and engage in entertaining activity within 3D distributed virtual environments. It originally uses its own network module which just provides a primitive network functionality without any specific network management or filtering schemes. We integrated ATLAS with VP by replacing the network modules with ATLAS as a short project over the course of two weeks. VP is written in Java. We use the Java version of ATLAS to integrate the two systems. The client/server model is used.

The integrated system exploits region-based filtering to reduce communication overhead. VP uses a portal concept to allow a user to move to another place. Each specific place, which is separated by a portal, is defined as a separate region. Since there is no interaction among regions, the inter-region interaction scheme is not used. However, the user interest group based filtering is used in a region. For concurrency control, we adopt the entity-centric prediction based scheme to VP to support real time interaction. The download time of the world data is improved by using the ATLAS user-based caching and prefetching method. Figure 28 shows a user navigating in the integrated system.

Another application, ICU/ETRI Virtual shopping mall (Han et al., 2002) is a web-based application which uses ATLAS as a communication infrastructure and RTV X3D viewer (www.realtimevisual.com) as a browser. The integration took about one month due to the definition of additional APIs and adjustment of ATLAS to the application.

Based on the client-server model, the application consists of ATLAS servers, a web server, and clients. The ATLAS servers divided into a session server and region
servers manage a consistent state of a virtual world, interest, concurrency control, data distribution of users, and load distribution among servers. The web server performs distribution of static virtual world data, avatar information, product information, and world rule information for dynamic objects in a virtual world.

A client, written in C++, is composed of an ATLAS client module and a RTV X3D browser which is an ActiveX control. Communication and event passing between an ATLAS client and an X3D browser is done using EAI (Carey, Bell, & Marrin, 2001) and the extended interfaces of the browser.

To enter a virtual shopping mall, users first connect to the web server, and download an applet and an X3D browser to set up a user interface. As a user selects a session and an avatar type which the user is interested in, the static world data and world rule information are downloaded from the web server in order to initialize the virtual world. Using the world rule information, an ATLAS client initializes dynamic objects and their properties obtained through EAI. To support latecomers, the client downloads the current state information of dynamic objects from the ATLAS server and updates the virtual world.

To support scalability, we divide a virtual shopping mall into several logical regions. The application then applies the user interest group based filtering scheme and the inter-region interaction scheme together as interest management. For real time interactive concurrency control, we use the entity-centric prediction based scheme rather than the pessimistic one. We also adopt the user-based caching and prefetching method to the application to reduce download time of world data from the server. During operation of the virtual shopping mall, the ATLAS server performs load distribution when it becomes overloaded by a lot of highly skewed users. Figure 29 shows a client view of the virtual shopping mall.

3.4.3 ATLAS + 3D Multi-User Game. CyRun is a treadmill virtual space developed by Gasiopeia Company (www.gasiopeia.com). As a user runs on the treadmill, the CyRun application represents the user as an avatar running in a virtual space. Since the application is developed for a single user, we have extended it to a multi-user application in which users running in the same virtual space can share their views and play a virtual marathon with each other. The integration took about two months and we used most of the time tuning the performance and stabilizing the system for commercialization. Based on the peer/server model, the CyRun server integrated with the ATLAS server manages sessions, regions, and users. The CyRun client embedded in the treadmill is integrated with the ATLAS client and multicasts update messages of the user running on the treadmill.

The application can use various interest management schemes to reduce the number of messages as the number of users increases. However, since a user only runs on the treadmill and does not interact with other objects, the system does not need any concurrency control. For data replication, we can easily use the user-based caching and prefetching scheme because a user always runs along the predefined route and the user’s behavioral pattern is fixed. Figure 30 shows an image capture of the CyRun display.
4 Conclusion

Scalability is one of important design issues in DVEs. In this paper, we analyze the scalability issues in terms of communication architecture, interest management, concurrent control, data replication, and load distribution. We propose ATLAS, a network framework for DVEs that supports scalable solutions in these five aspects. ATLAS provides a set of APIs in Java and C++ that suits various application requirements. To meet various application requirements, we support a peer/server model as well as a client/server model. To improve scalability, ATLAS allows a user to receive update messages only from others in whom the user is interested instead of all users in the same and neighboring regions, which enhances the interactive performance. To resolve conflicts of concurrent updates of objects and grant ownership to the right user in time, ATLAS provides a prediction-based concurrency control scheme in which the current owner of an object receives ownership requests only from users adjacent to the object, not from all users in the same region. When replicating objects from a server at local hosts, ATLAS reduces downloading time by caching and prefetching only objects in which users are interested, in terms of proximity between users and objects, and access priority of objects based on the user’s behavior pattern. For distributing the workload of overloaded servers to less-loaded ones, ATLAS selects a set of servers involved in load distribution by dynamically adapting to the workload status of servers, which keeps the interactive performance of users acceptable even though users are not uniformly distributed over the virtual environment. Successful integrations of ATLAS with several applications ensure its versatility.

To support diverse requirements of the users, we currently work on extending ATLAS to allow a DVE system to be dynamically extended or adapted to new services during runtime. While existing approaches (Watsen & Zyda, 1998; Oliveira, Crowcroft, & Slater, 2000) mainly focus on dynamic extensibility, which provides a base framework to dynamically plug in various components at runtime, we focus on self-tuneability which enables ATLAS to automatically configure itself, adapting to the current user requirements, execution environment, or network state (Han & Lim, 2003). We are also investigating how ATLAS servers can dynamically share the load with others in the way of distributing the tasks of one server to others (Lim & Lee, 2004).

Acknowledgments

We would like to thank members of CDSN laboratory for their work and comments during the development of the system. We would also like to thank K. Wohn and H. S. Kim at KAIST, Tom Furness and Bruce Campbell at HIT Lab, B. T. Choi at ETRI, and C. J. Park at Gasiopelia Company for their help with this work. This research is supported by Digital Media Lab, and the National Research Laboratory Program of Ministry of Science and Technology, the Republic of Korea (Grant No. M1-0318-00-0130).

References


