GROUP: Dual-Overlay State Management for P2P NVE

Eliya Buyukkaya*, Maha Abdallah*, Romain Cavagna*, Shun-Yun Hu**
*Laboratoire d'Informatique de Paris 6
University of Paris 6, France
{Eliya.Buyukkaya, Maha.Abdallah, Romain.Cavagna}@lip6.fr
**Department of Computer Science and Information Engineering
National Central University, Taiwan
syhu@csie.ncu.edu.tw

Abstract

Peer-to-peer (P2P) architectures have recently become a popular design choice to build scalable Networked Virtual Environments (NVEs). While P2P architectures offer better scalability than server-based architectures, efficient distribution and management of avatar and object states remains a highly challenging issue. In this paper, we propose GROUP, a fully-distributed P2P architecture for NVEs that addresses this issue by combining a structured P2P overlay, used for object state management, with a Voronoi-based overlay, used for avatar state and group membership management. The resulting dual overlay architecture enables efficient and fully distributed management of state updates for P2P-based NVEs.

1. Introduction

Networked virtual environments (NVEs) are computer-generated virtual worlds where users navigate and interact with their surroundings and each other through virtual representations called *avatars*. NVEs are traditionally supported by client/server architectures. The server (or a cluster of servers) keeps the world data, calculates the states and disseminates the state updates to users. A user informs the server about the events that she/he creates and is informed by the server about the states of other avatars and objects in the virtual world. However, centralized architectures can lead to high communication and computation loads at the servers, which quickly become a bottleneck point during peak operations. To overcome these problems inherent to centralized solutions, P2P overlay networks are emerging as a promising alternative for NVEs [2, 3, 6, 7]. In P2P systems, the overall system load is distributed among all participating users. By aggregating and sharing the users' resources, the system can achieve high scalability in a cost-effective manner.

A key aspect of P2P networks is the overlay topology that defines the way peers connect to others peers with whom they can interact and exchange messages. As many thousands to even millions of concurrent users may exist in NVEs, each individual user often has only a limited view. How peers can most efficiently connect to each other to support proper interactions within the view is thus of central concern to P2P NVEs. Concepts such as *Area of Interest (AOI)* [10], *interest group* [7], *publisher / subscriber set* [11] have also been used to identify the set of peers to connect or to interact.

In this paper, we propose *GROUP*, a dual-overlay P2P architecture for NVEs. In our system, the set of interacting peers is based on the concept of *group*, which defines a set of peers who are interested in the same entity. The term *entity* refers to either an avatar or an object in the virtual world. The *manager* peer of an entity keeps the entity's data, calculates its state updates and disseminates the updates to members of the group. Our architecture combines a structured overlay based on a dynamically partitioned rectangular grid with a second structured overlay based on a Voronoi diagram [1]. The grid overlay provides state management for the static object data, while the Voronoi overlay provides state management for the more dynamic avatar data, as well as membership management for the groups. The intuition behind this choice is to avoid avatar clustering that might overload a given manager by using a flexible overlay, while preventing unnecessary entity data transfer between managers by using a more stable assignment for the static objects. By considering entities separately, we can calculate and update entity states more efficiently. Our main contribution is thus a fully-distributed P2P architecture that provides a complete

This work is supported by the "Mad Games" project of the ANR RIAM program.

solution to the distribution and management of both avatars and objects in NVEs.

The rest of this paper is organized as follows. Section 2 describes prior work and our contribution. Section 3 defines our system model. Section 4 details our architecture. Section 5 provides a discussion of our solution. Finally, section 6 concludes this work and points out some future perspectives.

2. Related work

A number of P2P designs have been recently proposed to support scalable NVEs.

Colyseus [2] uses a traditional randomized DHT or a range-queriable DHT to discover objects for peers. One primary node for each object is responsible for the object's management, and any number of read-only object replicas may exist on the other nodes, enabling quick local access to the object. However, Colyseus's DHT-based architecture can lead to unacceptably long latency due to queries on the DHT. We take advantage of the logical connections between avatars/objects in the virtual world that Colyseus does not take into consideration.

In SimMud [7], the virtual world is partitioned into many fixed-size regions, each of which managed by a unique coordinator peer. The coordinator of a region also serves as the root of a multicast tree to which all peers within the region subscribe. State updates within a region are multicast by the root node to all region members. However, the coordinator can quickly become a bottleneck and suffer from the performance, scalability, and robustness issues faced by centralized architectures when user crowding occurs in a region.

In MOPAR [12], the authors partition the virtual world into fixed-size, continuous hexagonal cells, each of which is assigned a unique master node within the cell. A cell's master keeps track of all other slave nodes in its cell, and regularly exchanges this list with the master nodes of neighboring cells. Slave nodes are notified of new neighbors by their masters, while other message exchange is performed directly between the slaves. However, MOPAR considers only avatar-avatar interactions, but no object management.

Rieche et al. [9] use a CAN-based [8] approach to partition the world into dynamic zones. The players are assigned to the zone servers according to their positions in the world. Varvello et al. [11] propose to partition the virtual world into dynamic cells and participants adopt a publish/subscribe mechanism within cells. HYMS [6] also partitions the virtual world into multiple dynamic cells. A responsible central cell server exists for each cell. Multiple qualified clients can takeover server loads and communicate in P2P fashions among themselves.

The work initially presented in VON [5] and later extended in VSM [4] discusses a Voronoi-based [1] partitioning for NVEs. VON considers only position updates and does not deal with state management. In VSM, the authors further consider state management; however, the same algorithm used for peer updates in VON is used for the dissemination of object updates. This leads to situations where some peers are not informed about state updates for objects visible to them. As a peer p knows only about other peers inside its AOI, if any of p's objects is seen by peers that are not inside its AOI, then p cannot disseminate the object's state update to those interested peers.

Buyukkaya and Abdallah [3] proposed a data management algorithm in the context of Voronoi decomposition of the world. However, since object management is tightly coupled with world partitioning, more static objects in the virtual world object may be continuously transferred and reassigned between peers due to the high dynamism of the partitioning, creating significant overheads. This problem occurs for both the work in [3, 4].

The present work proposes a P2P-based NVE state management scheme, designed with all the above issues in mind, i.e., dynamic partitioning with considerations of the relative spatial locality relations, support of different entity visibilities, and efficient state dissemination.

3. GROUP: a dual-overlay architecture

We based our system on the fact that the peers can be grouped by the message transfer between them and that the message source can be the core of the group. Peers who are the receivers of message updates from the same message originator thus can be grouped together. We also observe that dynamic entities (e.g., user avatars) and static entities (e.g., objects such as trees and treasure boxes) in virtual worlds exhibit different dynamics and may best be treated separately. To manage groups for both dynamic and static entities, we adopt a *dual-overlay* approach. By combining a structured overlay based on a grid partitioning with a second structured overlay based on a Voronoi diagram, the management of static and dynamic entities can be separated. We thus avoid overloading a given manager due to avatar clustering by using a flexible overlay for avatars, and also prevent unnecessary entity data or ownership transfer between managers by using a more stable grid overlay for the static objects.

Our architecture is based on the concept of *group*, defined as the set of peers interested in the same entity. The term *entity* denotes either an *avatar* or an *object* in the virtual world. We distinguish between avatars and objects by both their dynamics and behavior logic. *Avatars* represent users, and are considered to be more dynamic.

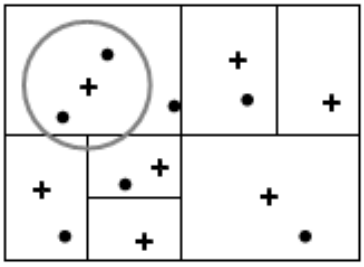


Figure 1. Grid overlay

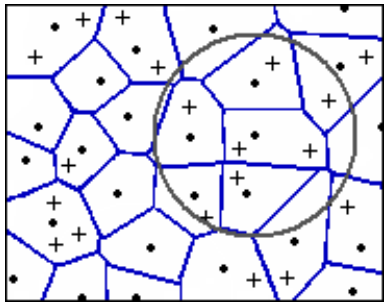


Figure 2. Voronoi overlay

They may generate *events* that would affect the states of nearby avatars or objects. *Objects*, on the other hand, are considered to be more static in positions, and are often passively affected by the actions of avatars.

An entity is visible to peers within a *potentially visible region (PVR)*. Our system supports arbitrary-shape PVR. However, in order to simplify presentation, we first assume that an entity has a circular PVR, the center of which is the entity's coordinate in the virtual world (Figure 1). We note that it is straightforward to generalize our approach for PVRs with different sizes, which may be useful for certain applications.

Each entity has a *manager* that keeps the entity states, updates its states and disseminates the updates to the group of interested peers (i.e., users who can potentially see the entity). To be a member of a group, a peer needs to subscribe to the entity's manager in order to receive the entity's updates. When a peer is no longer inside the PVR of an entity, it unsubscribes from the manager of the group. The discovery of new group members is done by the *boundary members* of the group, which are the outermost members inside the PVR.

3.1. Grid overlay

The grid overlay supports the management of object states in the system. The virtual world is first dynamically partitioned into rectangular grids (Figure 1), similar to a CAN [8]. Each grid is then assigned to an *object manager* peer, whose avatar coordinates do not necessarily reside in the grid. Each object is assigned to one object manager based on in which grid its coordinates fall. An object manager is responsible of managing all the objects within its grid. It keeps the objects' data, calculates new object states based on the current events of nearby entities, and collectively sends objects' state updates to concerning peers. Each object manager also keeps contacts with neighboring managers with whom it shares a common edge.

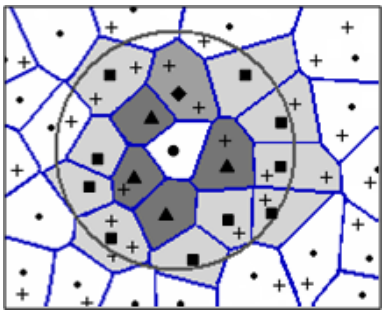
In Figure 1, the circle shows the PVR of the object residing in the center of the circle. The object is visible to two peers inside the PVR. These peers are members of the group managed by the object's manager.

In our system, object assignments to peers do not depend on the peers' positions in the virtual world, so frequent peer movements do not cause excessive object ownership transfers between peers. In addition, by assigning spatially nearby objects to the same object manager, the object's current states can be calculated and distributed efficiently in aggregations (i.e., a peer who sees and subscribes to several nearby objects can receive updates from the same manager).

3.2. Voronoi overlay

The Voronoi overlay mainly supports the state management for avatars in the system. An avatar is assigned to a manager based on the Voronoi overlay, which also contributes to the management of group membership for objects.

The structured overlay is based on a Voronoi diagram [1], which partitions the virtual world into many Voronoi regions, each of which is managed by a peer, whose coordinates in the virtual world determine the Voronoi partitioning (i.e., each peer lies in the region that contains all the points that are closer to it than to any other peer) (Figure 2). The reason behind the use of the Voronoi partitioning is that it provides locality by connecting each node with its geographically closest neighbors in the plane, thus enabling direct message exchange between nodes that could potentially interact. This locality feature achieves high scalability by limiting the number of neighbors that a node is connected to, independently from the size of the network. A global behavior is then achieved through cooperative local interactions. Voronoi partitioning also allows distributed state management, by partitioning the virtual world into regions equal to the number of nodes, management tasks can become fairly distributed.



▲ Direct neighbors ■ Boundary members
⧫ Direct neighbors & Boundary members

Figure 3. Peer relations

Each Voronoi region contains only one avatar but may contain several objects. The *avatar manager* is the peer that manages the avatar's group. It maintains and performs state management for the avatar's data. As mentioned in the previous section, an object manager is responsible for the objects residing in a grid. However, as objects may also reside inside a Voronoi region, the *avatar manager* thus also keeps information about the object managers for the objects within its Voronoi region with no empty groups (i.e., no peer exists within the objects' PVR). The objects with empty groups need to be kept by the avatar managers because the avatar manager is the closest peer to those objects (i.e., it is the most probable node to become the first member in the object's group). Avatar managers thus should know the object's object manager in order to receive the object's updates when the time comes.

In the Voronoi overlay, there are two types of relations between peers: *direct neighbors* of a peer and *boundary members* of an entity group. For a given peer p, a peer q is called *direct neighbor* if there is a common edge between the Voronoi regions of p and q. A peer r is called *boundary member* of an entity group if r is one of the outermost peers of the group, more formally, if (1) r lies inside the PVR of the entity; and (2) at least one direct neighbor of r lies outside the PVR. Figure 3 shows the direct neighbors of a peer and also the boundary members of its group.

4. State management with GROUP

4.1. Entity assignment

Each avatar is assigned to the peer whose user controls the avatar (i.e., the peer inside each Voronoi region is the *avatar manager* for the avatar's group). Respectively, each object is assigned to the peer that manages the grid within which the object resides (i.e., the peer assigned for each grid is the *object manager* for all the objects inside the grid). A grid (and hence its objects) is assigned to a peer by using a consistent hash function to map the peer identifier (which includes IP address and port number) into the grid space, of which the peer takes over the responsibility. A peer thus can be both the manager of its avatar and the manager of the objects of its assigned grid.

Our system keeps the relative spatial locality between entities by using a grid partitioning to assign object ownerships instead of a random distribution. Message aggregations are thus possible during update disseminations. By distributing objects among peers according to a more static grid partitioning, the system avoids unnecessary object data or ownership transfer between peers (i.e., object ownerships do not change easily due to avatar positions, as required for [3, 4]).

4.2. State management

State management deals with 1) the generation of events, 2) the processing of events, 3) the modification of states, and 4) the dissemination of updates. If a group member of an entity creates an event that could affect the entity, the member should send the event to the entity's group manager. The manager then calculates updates for the entity's states based on the entity's current states, the received event, and the current states of nearby objects. So, the final state of the entity is determined by the manager. When the entity changes its state, the manager sends the update to the group members.

Our system also allows peers to insert objects into the virtual world, by informing the object manager of the grid region.

4.3. Member discovery

When a member of the entity group receives a position update for the entity or its direct neighbors, it checks if it has become a boundary member or if it is no longer inside the entity's PVR. In the latter case, it sends an unsubscription request to the entity manager. This way, the manager is able to detect obsolete members and remove them from the group.

The discovery of new members is done through the boundary peers of the group. When a boundary member of an entity's group receives a position update of the entity or its direct neighbors, it checks if one of its direct neighbors has entered into the entity's PVR. If a new member peer is found, it informs the new peer (i.e., its direct neighbor) about the entity's manager. The peer then sends a subscription request to the manager. This way, the manager detects newly entered peers within the entity's PVR and

adds them to the group as members. The member peer then starts to receive the entity's updates through the manager.

4.4. Movement

PEER MOVEMENT. When a peer p moves, it informs the members of its avatar's group about its position update. Since the boundaries of the Voronoi region have changed, some objects may now lie in the regions of p's direct neighbors. If an object has moved into a nearby region, but whose avatar manager is not a member of the object's group, p would inform its direct neighbor about the object's object manager, so that this direct neighbor can inform the object manager about the change in the avatar manager. Likewise, p is also informed by its direct neighbors about any newly entered objects into its region.

OBJECT MOVEMENT. In case of an object o's movement, if o now resides in a different Voronoi region, its avatar manager then becomes the peer who resides in the new region. If o's group is empty (i.e., no peer exists within o's PVR), the new avatar manager would inform o's object manager about this responsibility change. Additionally, if o now lies in a different grid, then o's object manager becomes the peer assigned to this grid. o's previous manager should transfer o's data to the new manager. Afterwards, o's new manager becomes responsible for the update dissemination to group members. If o's group is empty, the new object manager informs the object's avatar manager about this responsibility change.

4.5. Peer arrival/departure

Entrance to the virtual world is done through *gate peers*, which are a set of peers whose IP addresses are well-known in the system. A joining peer p first contacts one of the gate peers to join the Voronoi overlay. The contacted gate peer then identifies an existing peer q whose Voronoi region contains p's joining position by forwarding p's entry request towards the intended join position. Peer q then informs p about p's direct neighbors and the members within p's PVR. p in turn constructs and initializes its avatar group based on the group member lists of its direct neighbors. p also takes over the responsibility of the objects residing in its region. For objects without any members, p informs the objects' object managers about this responsibility change.

For the new peer to join the grid overlay, a consistent hash function is first used to map its identifier (which includes IP address and port number) into a grid space, of which it takes over the responsibility. The previous object manager of the grid would transfer the objects' data to the new peer to continue the dissemination of object updates to group members. For an object with no members, the new peer informs the object's avatar manager about the change in object managing responsibility.

In case of a peer p's departure, one of p's neighbors in the structured overlay takes over the responsibility to manage objects in p's grid. In the virtual world, p unsubscribes from the groups of which it is a member. The responsibility of objects with no members inside p's Voronoi region is taken over by p's direct neighbors.

5. Discussions

Our architecture is fully distributed since the full requirements for state management are distributed to all peers; even the entrance to the system is done via a number of gate peers. No bottleneck point thus exists in the system as client-server architectures do. Note that our design accommodates both homogeneous and heterogeneous peer capacities. By simply allowing all peers to be object managers, or only some selected super-peers as object managers, the system can flexibly adjust the degree of resource utilization for peers. Our solution is scalable since each peer has only a limited number of connections to other peers, independent from the scale of the system. Game state consistency is also maintained with low latency, as an entity manager connects directly to its group members, where updates regarding the entity states can be sent to each group member in only one communication hop.

Although the issue of peer failures is out of our current scope, a simple replication can be used to deal with this problem. In face of a peer failure, the system could lose the peer's avatar data, the information of the objects with empty groups inside the peer's Voronoi region, and the object states managed by the peer within its grid. If we assume that the system does not need the avatar data during the avatar's absence, then for the other two types of data: 1) in the grid overlay, we could replicate the objects located in the peer's grid at one of the grid neighbors, the object data can thus be recovered from the grid neighbor; 2) in the Voronoi overlay, a peer's failure is first detected by its direct neighbors, who could recover the manager information of the objects with empty groups through its knowledge the object managers of those objects.

6. Conclusion

In this paper, we have described GROUP, a fully-distributed P2P state management scheme for networked virtual environments. Our solution exploits a dual overlay design to distribute world data evenly among all the participating peers. The grid overlay provides state management for static object data, while the Voronoi overlay provides state management for dynamic avatar

data, as well as membership management for the groups. The flexible Voronoi overlay prevents the overloading of a given manager caused by avatar clustering, while the more stable grid overlay avoids unnecessary entity data transfer between managers for static objects. This enables both the calculations and updates of entity states to be done more efficiently. GROUP is thus a fully-distributed P2P architecture that provides a complete solution to the distribution and management of both avatars and objects in NVEs. For future work, we plan to evaluate our algorithm with both simulations and real-world settings.

References

[1] F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.

[2] A. Bharambe, J. Pang, and S. Seshan. Colyseus: a distributed architecture for online multiplayer games. In *Proc. ACM/USENIX 3rd conference on Networked Systems Design & Implementation (NSDI)*, pages 12–12, San Jose, CA, USA, May 2006.

[3] E. Buyukkaya and M. Abdallah. Data management in Voronoi-based P2P gaming. In *Proc. of IEEE International Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology (CCNC)*, pages 1050–1053, Las Vegas, NV, USA, January 2008.

[4] S.-Y. Hu, S.-C. Chang, and J.-R. Jiang. Voronoi state management for peer-to-peer massively multiplayer online games. In *Proc. of IEEE International Workshop on Networking Issues in Multimedia Entertainment (CCNC)*, pages 1134–1138, Las Vegas, NV, USA, January 2008.

[5] S.-Y. Hu, J.-F. Chen, and T.-H. Chen. VON: A scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4):22–31, July/August 2006.

[6] K.-C. Kim, I. Yeom, and J. Lee. HYMS : A hybrid MMOG server architecture. *IEICE transactions on information and systems*, 87(12):2706–2713, December 2004.

[7] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proc. IEEE INFOCOM*, pages 96–107, Hong Kong, China, March 2004.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. Conference on Applications, technologies, architectures, and protocols for computer communications (ACM SIGCOMM)*, pages 161–172, San Diego, California, USA, 2001.

[9] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak, and G. Carle. Peer-to-peer-based infrastructure support for massively multiplayer online games. In *Proc. IEEE Consumer Communications and Networking Conference (CCNC)*, pages 763–767, Las Vegas, NV, USA, January 2007.

[10] S. Singhal and M. Zyda. *Networked virtual environments: design and implementation*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[11] M. Varvello, E. Biersack, and C. Diot. A networked virtual environment over KAD. In *Proc. ACM CoNEXT conference (CoNEXT)*, pages 1–2, New York, NY, USA, December 2007.

[12] A. P. Yu and S. T. Vuong. MOPAR: a mobile peer-to-peer overlay rrchitecture for interest management of massively multiplayer online games. In *Proc. International workshop on Network and operating systems support for digital audio and video (NOSSDAV)*, pages 99–104, Skamania, Washington, USA, June 2005.