Kiwano: A Scalable Distributed Infrastructure for Virtual Worlds

Raluca Diaconu
Laboratoire d'Informatique de Paris 6, UPMC
Email: raluca.diaconu@{lip6.fr, orange.com}

Joaquín Keller
Orange Labs, Issy-les-Moulineaux
Email: joaquin.keller@orange.com

***Abstract*—In today's virtual worlds, even in the massively multi-player games, the number of avatars together in a virtual place barely reaches thousands. Regarding the billions of users dwelling the web, this is astonishingly low. In this paper we identify spatial indexation as the bottleneck that impedes scalability and we propose Kiwano, a distributed system to scale up to millions and more. In order to split the load, Kiwano dynamically divides the space in zones, each having a server maintaining a spatial index of all moving objects within the zone. This allows to timely compute which objects are in the neighborhood of any avatar. Thus, knowing its immediate environment, the client is able compute the local scene.**

Keywords: Scalability, Virtual Worlds, Voronoi Tessellation

INTRODUCTION

Internet hosts services accessed by billions of users, supported by gigantic datacenters of hundreds of thousands of processors. However, surprisingly, virtual worlds are still unable to host more than few hundred simultaneous users in a same contiguous space [4], [3].

To cope with many users actual solutions abandon crucial features and divide the space in many tiny pieces. For example, in World of Warcraft users are distributed among hundreds of separate "realms" while Second Life is a patchwork of thousands of mostly empty "sims".

To achieve massive scalability, we propose Kiwano, a novel distributed system using a self-resizing zones approach. The zones adapt their shapes according to the Voronoi tessellation of the positions of the avatars and within each zone the corresponding dual Delaunay triangulation serves as spatial index. Servers, each in charge of a zone, are communicate in a peer-to-peer network to reshape their zones.

The spatial index allows a fast extraction of which moving objects are in the neighborhood. The zone servers can then notify timely the avatars of the relevant events occurring in their vicinity.

I. DRAWBACKS OF EXISTING SOLUTIONS

To accommodate the millions of users of Massively Multiplayer Online Games (MMOGs) different solutions have been implemented.

Fixed zones: To split the load between servers, it seems natural to partition the territory in contiguous non-overlapping zones, each zone being populated with less avatars than the maximum manageable by one server. Zone servers receive the position updates of the objects within their zone and inform the avatars of the modifications occurring in their proximity. This approach is very efficient when object distribution remains the same over time so each zone is in charge of a roughly constant number of objects.

But in actual systems most regions are empty while 1% are overloaded [6]. This is highly inefficient as the resources allocated for the empty regions remain unused. Moreover large groups are still impossible. For instance, in Second Life meetings gather less than a hundred simultaneous avatars.

Sharding/instancing: Another solution [3] to accommodate many users is to run instances, called "shards," of the same virtual world. Avatars in two different shards may have similar experiences but since they cannot interact they are not actually in the same place.

Peer to peer: Another option to increase scalability is to use a peer-to-peer architecture [5] where clients carry out functions usually devoted to servers. Despite the amount of research carried on, these solutions have failed to be adopted by the industry. To explain this many reasons [6] are invoked. One of the most prominent is the weak performances of these systems which are degraded by the capacities the slowest peers.

II. KIWANO

In the physics of most virtual worlds it is assumed that events have only a local, limited effect. Therefore, for each avatar there is a neighboring zone such that knowing it suffices for the computation of the complete scene in its field of vision. Thus avatars must be notified of all the events produced by objects located in their neighborhood.

Our proposed model sets apart the static and the dynamic aspects of the scene. The decor or static objects are handled by classical scalable spatial databases such as [2], while Kiwano takes care of the moving objects. We have designed Kiwano to support an unlimited number of moving objects updating their position at arbitrary high frequencies.

The main drawback of the fixed zone approach is the uneven load attribution. But, for a given distribution of the moving objects, by choosing properly the shapes of the zones, a good load balancing can be achieved. However to keep the balancing as distribution changes, zones should dynamically adapt. To do so, in Kiwano zone shapes are determined at any moment by the positions of the objects.

A. Zones

A zone is composed by the Voronoi cells of the points representing their inner objects. Zones do not overlap but the objects at the frontier of the bordering zones are used to compute the tessellation. The composition of the Voronoi tessellations associated to each zone produces the Voronoi tessellation of the entire set of positions.

To maintain contiguity and to equilibrate the load, the set of objects attributed to a zone changes. Within each zone the tessellation is also updated when objects move. Thus the zones reshape dynamically.

The dual Delaunay triangulation is used to determine neighbors. In Kiwano we define a family of neighborhood relations: for a every n, two object are said to be neighbor^n if there are at most n hops in the Delaunay triangulation.

Virtual worlds are usually meant for interaction with others (chatting, gaming, etc...). They mimic the real world and interactions only occurs between neighboring entities. For humans, interaction is limited in the number of simultaneous participants.

With neighbor^3 avatars have an average of 70 visible neighbors which should be enough for most scenarios. Henceforth, in this paper the neighborhood is defined as neighbor^3.

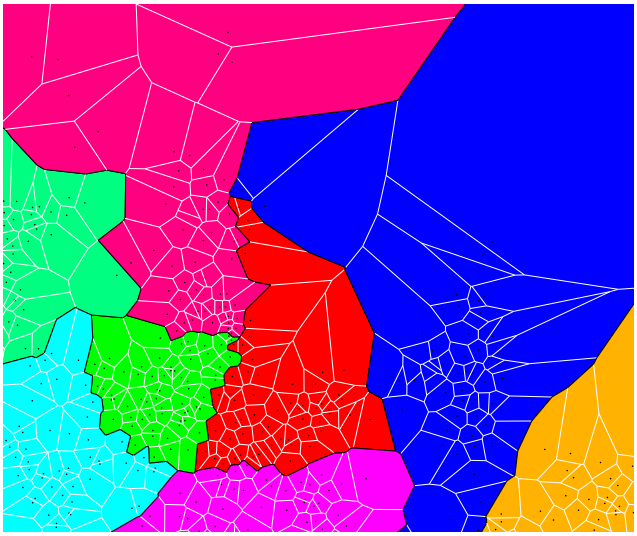

Fig. 1. Example of dynamically generated zones at a given moment

B. Architecture

Each zone is managed by one server. Zone servers are organized in a peer to peer network. A server communicates with the servers of the neighboring zones using the zone border protocol. This protocol provides a self-stabilizing procedure for shaping the border to ensure load balancing. By propagating events occurring on the border it also provides awareness for objects that span their neighborhood on multiple servers.

An object sends the position to and receives the notifications from one server at a time. Notifications are sent when objects move, arrive and leave the neighborhood. Events as avatar animation, chat, or any application specific events occurring in the neighborhood are notified at proxy level.

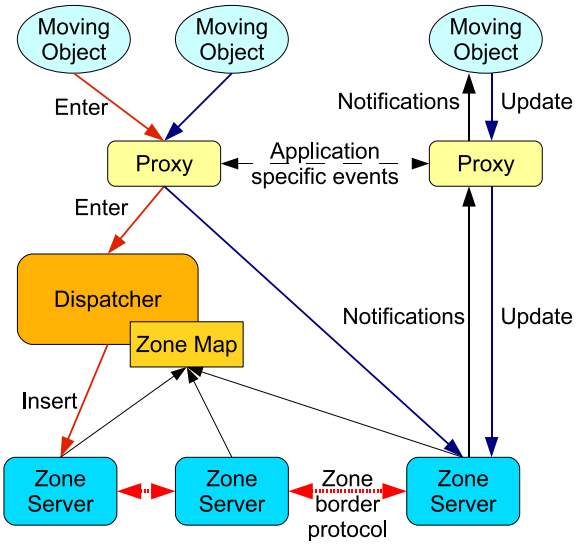

Fig. 2. Kiwano architecture

The dispatcher assigns a initial zone server to objects when they join the system. It selects the server using a zone map regularly updated by the servers in the network.

While they are present in the world, the moving object may switch zone servers. To make this transparent to the developer, Kiwano provides proxies. For an avatar, the proxy remain the same during the whole session.

Altogether, Kiwano hides the distributed nature of the system.

C. Algorithm

Events in the system are generated by the moving objects as they enter the virtual world or update their position, and by the zone servers. A zone server requests insertion, when joining the network, forward position updates as objects at the border move, and regular-time interval updates to maintain the zone map.

Object insert: When joining, a proxy is provided to the new object. It sends to the dispatcher an insert query with its own id and object's first position. The role of the dispatcher is to find the best entry server for the new object in the network of zone servers. The dispatcher uses the *zone map* to select a zone containing the position with great probability.

The dispatcher forwards the query to the chosen entry server. If the new object's position is inside the zone, the right server have been found. The object is inserted in the zone and the neighboring zones are recomputed if needed. Otherwise, the server forwards the query to a bordering zone server. This server is chosen by finding the nearest object on the border.

Finally, the matching server informs the proxy where to send future position updates and receive notifications. It sends a first notification with the list of all moving objects currently in the neighborhood of the newly inserted object.

Object position update: The proxy sends the update to the zone server. In addition to updating the internal spatial index with the new position, the zone server is responsible for notifying the neighboring objects affected by this movement.

In case the object was on or is moving to the border, zone shape may change and the bordering servers must be informed of the update. If the object crosses the border and goes out of the zone, it is removed from the current zone and reinserted as described above.

We make use of the nice properties Voronoi diagrams exhibit: an object movement affects only the surrounding neighbor cells of the old and the new positions.

Algorithm 1 on zone server: update object's position

```
store old neighbors and old bordering servers
update object's position and the Voronoi diagram
for all old neighbors and new neighbors do
    update list of neighbors and notify
if new position is out of zone then
    insert the object to the nearest bordering server
for all old bordering servers not in new bordering servers
    send message to remove object from its border
for all new bordering servers not in old bordering servers
    send data to add object to its border
```

Object removal: When an object leaves, the server runs the same algorithm as for position update but considers the set of new neighbors empty. Consequently, the proxy in charge of the object dismisses it.

Server insert: When joining the peer to peer network of zone servers, a new server must be provided with a minimal zone and its bordering servers. The smallest possible zone has one moving object.

When connecting, a new zone server Z_i sends a request to the dispatcher. The dispatcher chooses an entry point in the server network, Z_j, preferably the most loaded one to help reducing its load. Z_j selects an object on the border and inserts it into Z_i, along with the information about its neighbors. By knowing the neighbors, Z_i is able to compute its initial zone as the Voronoi cell of the unique object and initiates the communication with the bordering servers. The inter-zone load balancing procedure will add up more objects.

Zone map update: To select the right zone server when inserting an object, the dispatcher needs to be aware of the precise map of the zones. But the shape of a zone can be a complex polygon with up to $\mathcal{O}(n)$ vertices, where n is the number of objects in the zone. And since zones reshape frequently, we have chosen instead to represent each zone by a single point. We investigate two policies to choose this point.

a) Zones are represented by the position of a sample object. The dispatcher receives position updates when the sample object moves or when the server selects a different one.

b) Each zone server Z_i with n objects and V_i the set of indexed objects is represented by its barycenter c_i whose coordinates are incrementally updated as follows:

$$x_{c_i} = \frac{1}{n}\sum_{p \in V_i} x_p, \qquad y_{c_i} = \frac{1}{n}\sum_{p \in V_i} y_p$$

At regular time intervals or when c_i considerably changes position, Z_i sends the actual position to the dispatcher.

Server withdrawal: Before a zone server leaves it must make sure each object inside its zone is taken by another server. The server shrinks the covered area by steadily passing to its neighbor servers the charge on the border. The procedure is repeated until the covered zone is empty. In case of failure, the proxies ask the dispatcher for reinsertion of the affected objects.

Load balancing: As objects move, leave or join the system, the set of objects and the covered zone for a given server change. In order to balance the load, servers communicates their load to the bordering servers at regular time intervals. When the load of two bordering servers greatly differs, the two servers exchange objects close to the frontier. Consequently, the border moves toward the most loaded server, adding up to the other. As this local procedure is repeated, the load distributes among the entire network of servers.

III. Conclusion and future work

We hope with Kiwano to be able to host an unlimited number of simultaneous users in a contiguous virtual world.

Kiwano's notion of awareness might not be compatible with the physics of some virtual worlds. For instance events with effects spanning over large regions are not handled efficiently by Kiwano. However for most applications proximity interactions are satisfactory.

The next steps are to run simulations to verify the system and evaluate performances. Particular attention should be paid to flash crowds in the world. For other uses, we are working on spatial queries using the Kiwano's distributed index.

Kiwano has already raise great expectations and contributes to a project aimed at building a scalable Minecraft world. Also, an ongoing project within our team at Orange Labs, Hybrid Earth, is build on top of Kiwano. Hybrid Earth [1] enacts a mixed reality world at planet scale using Google Streetview and augmented reality on smartphones and specialized goggles.

Acknowledgements

Many thanks to Mathieu Valero of Orange Labs and Sébastien Monnet and Pierre Sens of LIP6.

References

[1] Hybrid Earth: Hybrid Reality at planet scale. *www.hybridearth.net*.

[2] James C. et al Corbett. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12. USENIX Association, 2012.

[3] Nitin Gupta, Alan Demers, Johannes Gehrke, Philipp Unterbrunner, and Walker White. Scalability for Virtual Worlds. In *International Conference on Data Engineering (ICDE)*, pages 1311–1314, 2009.

[4] Joaquin Keller, Raluca Diaconu, and Mathieu Valero. Towards a Scalable Dynamic Spatial Database System. 2012.

[5] Joaquin Keller and Gwendal Simon. Toward a Peer-to-Peer Shared Virtual Reality. In *International Conference on Distributed Computing Systems*. IEEE Computer Society, 2002.

[6] Amir. Yahyavi and Bettina Kemme. Peer-to-peer architectures for massively multiplayer online games: A survey. *ACM Computing Survey*, 2013.