Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

KIWANO: SCALING VIRTUAL WORLDS

Raluca Diaconu

Computer Laboratory
University of Cambridge
15 JJ Thomson Avenue
Cambridge CB3 0FD, UK

Joaquín Keller

Distributed Systems
Orange Labs Research
38-40, rue du Général Leclerc
Issy-les-Moulineaux F-92794, FRANCE

ABSTRACT

Kiwano is a distributed system that enables an unlimited number of avatars interact in the same virtual space. By separating the management of virtual world components –avatars, moving objects from the static décor– we take a novel approach and introduce a neighborhood relation between avatars. In Kiwano we employ Delaunay triangulations to provide each avatar with a constant number of neighbors independently of their density or distribution. The avatar-to-avatar interactions and related computations are then bounded, allowing the system to scale. The optimal number of avatars per CPU and the scalability of our system have been evaluated simulating tens of thousands of avatars connecting to an open Kiwano instance deployed across several data centers, in the cloud. These results exceed by orders of magnitude the performances of current state-of-the-art.

1 INTRODUCTION

Virtual worlds attract millions of users which routinely access them. However, they are still unable to host simultaneously more than a few hundred users in the same contiguous space. Since the early 1970s, when the first multi-user graphic virtual world appeared, the algorithmic complexity of running virtual worlds is $\mathcal{O}(N^2)$, where N is the number of users that are together in the same region (Debeauvais, Valadares, and Lopes 2012). In particular, when an avatar joins the world, all other avatars must be informed. Thus, the algorithmic complexity is $\mathcal{O}(N^2)$.

Contribution. To enable scalability we rely on the *separation of concerns* in virtual worlds, i.e., avatars, mutable objects, and static terrain. This allows us to propose a specific solution for *avatar scalability*. Our main contribution is Kiwano, a distributed system that *enables an unlimited number of avatars to be and interact in the same contiguous space.* In Kiwano we employ Delaunay triangulations to provide each avatar with a bounded number of neighbors independently of their density or distribution. The avatar-to-avatar interactions and related computations are then bounded, allowing the system to scale. The load is constantly balanced among Kiwano's nodes which *redistribute their load dynamically*. The optimal number of avatars per CPU and the performances of our system have been evaluated simulating tens of thousands of avatars connecting to a Kiwano instance running across several data centers. The results of these simulations are presented in this paper and they confirm the scalability of Kiwano: the needed resources per avatar remain constant when the number of avatars grows. These distributed algorithms and this architecture, fully described here for the first time, have been successfully used to scale a popular MMOG, Minecraft (Diaconu, Keller, and Valero 2013), (Valero, Diaconu, and Keller 2013), and to build a mixed reality world, HybridEarth (de Campredon, Diaconu, Keller, and Triponez 2014). We also designed a scalable architecture for existing virtual worlds such as Second Life (Diaconu and Keller 2014). With Kiwano our intention is to provide the first massively distributed and self-adaptive solutions for virtual worlds suitable

to run in the cloud. Kiwano is easily accessible via a public API available at `http://kiwano.li`, which enables *interoperability* between all connected virtual worlds.

Roadmap. We review some relevant works in Section 2. We introduce our notion of neighborhood based on Delaunay graphs in Section 3. Kiwano distributed algorithms are described in Section 4 and how they are deployed in cloud facilities for actual applications is explained in Section 5. We emphasize the cloud and interoperability features. The settings and measurements of our evaluation are in Section 6.

2 RELATED WORK

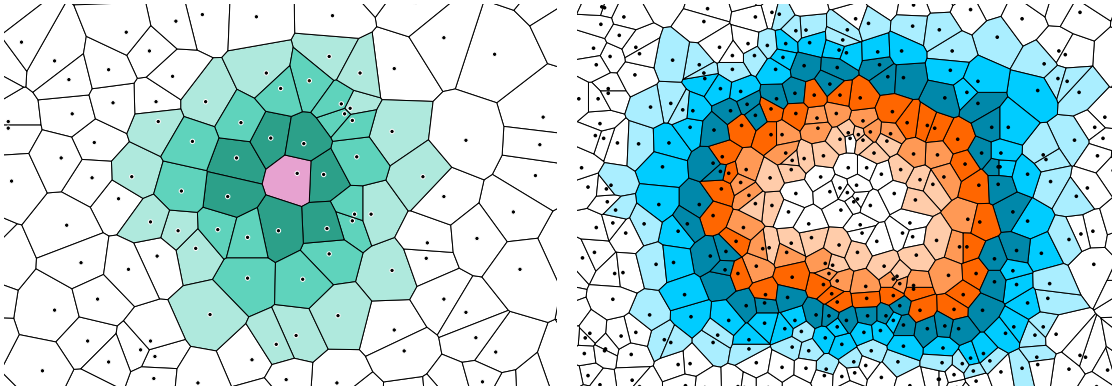
When the number of avatars and objects increases, it becomes infeasible to run a centralized world because (1) each user needs to handle more and more events of the virtual world and (2) the simulator reaches its limits in numbers of avatars and frequency of updates. To address (1) virtual worlds assume spatial locality: events have only a local, limited effect (*interested management*). For (2) the load must be *distributed*. Locality has thus far been defined in terms of space distance. Events are perceived within a circle around their location. Avatars receive updates of events happening in some area surrounding their position. As far as we know, locality has not yet been approached based on a neighborhood relation graph.

Interest management. Actual notions of interest management define, under various names, an *area* of interest. Interaction between avatars is possible if they are in each other's *area*, hence the geographical dependence. The limitation of fixed zones is that the avatar distribution is skewed, some areas are overcrowded while others are empty (de Oliveira and Georganas 2003), (Yahyavi and Kemme 2013). In VELVET (de Oliveira and Georganas 2003) Alice may see Bob while Bob may not see Alice when her area is larger. In VON (Hu, Chen, and Chen 2006) Alice and Bob maintain the connection if at least one can see the other. Donnybrook (Bharambe, Douceur, Lorch, Moscibroda, Pang, Seshan, and Zhuang 2008) proposes a fixed size *interest set* of other avatars to whom the player is paying attention. These are frequently updated, while all others, once per sec. It attains 900 simultaneous players in a p2p infrastructure.

Delaunay triangulations. And the dual Voronoi diagrams have been widely employed for proximity dissemination in *peer-to-peer systems* and are recognized as scalable (Keller and Simon 2002), (Hu, Chen, and Chen 2006), (Buyukkaya, Abdallah, and Cavagna 2009). Each node maintains a local triangulation of its position and the neighbors' positions. When avatars move in the virtual world they connect and exchange information from hop to hop. Similarly, MOPAR (Yu and Vuong 2005), (Knutsson, Lu, Xu, and Hopkins 2004) is a *hybrid* architecture that uses the best equipped computers (superpeers) to maintain fixed triangulated zones.

Architectures. All commercially successful virtual worlds are built upon *server-based* architectures (Minecraft, Second-Life, World of Warcraft). Multi-server solutions rely on space division. The cost of the computation is quadratic inside the region, and depending on their complexity, virtual worlds report different upper limits: $\sim$100 users/region in SecondLife, a few thousands for Minecraft (Diaconu, Keller, and Valero 2013), $\sim$120 users per shard for World of Warcraft. EveOnline (Hamburger 2013), reporting 3000 users, relies on particular features that allow time dilatation. This is not applicable in all scenarios. These suffer from a well-known problem, avatars want to join zones that are populated (Varvello et al. 2011) and finally, do not scale. Also, the world is not contiguous. Pikko Server (Almroth 2010) developed an optimized dynamic space partitioning algorithm that permanently re-evaluates the boundaries. It transfers players dynamically between game servers to avoid overloading. It established in 2012 a world record with thousands of concurrent users in a FPS battle. Pikko is a distributed, self-adaptive architectures. However, transferring chunks of the virtual world is costly, so it runs on a single (powerful) machine.

Some systems do a separation of concerns such as client management, physical simulation, script processing, and scene persistence (Lake, Bowman, and Liu 2010). Components run as independent services and are connected through the shared "scene graph." They reported support for up to 1,000 simultaneous avatars in OpenSim for a slightly reduced frequency of updates.



(a) Levels of neighborhood (b) A zone in Kiwano

Figure 1: Delaunay 3 data structure.

3 AVATAR INTEREST MANAGEMENT

The main idea is to separate the management of the three components –avatars, objects, and terrain. For each, the notion of area of interest changes. Although they must be located in the proximity of the concerned avatar, the way we pay attention to these components differs. The interest in avatars is for social interaction, objects may be tools, and the décor is fixed. This is the reason we employ a graph notion, of *neighboring avatars*. In Kiwano users can solely see and interact with their *neighbors.* The *neighborhood relation* is designed such that the number of neighbors remains within a range regardless of the avatar distribution.

3.1 Neighborhood relation

The notion of *neighbor* evokes (some sort of) proximity. In this section, we abstract the notion of *neighborhood relation* employing graphs where avatar positions constitute the vertices and possible interactions represent edges. In our approach we consider a graph with a degree of roughly fixed size, build on top of the Delaunay triangulation.

In what follows we use 2D Delaunay triangulations. In most virtual worlds long/lat coordinates suffice for neighborhood discovery (occasional cases where avatars are flying or climb are tolerated). Moreover, for N vertices, the size of the 2D Delaunay graph is $\mathcal{O}(N)$, while in 3D is $\mathcal{O}(N^2)$. Furthermore, we describe a *family of neighborhood relations* based on the *Delaunay power graph*, allowing us to chose the most suitable in terms of details vs. performance.

3.2 Delaunay power graphs

We note D or D^1 the Delaunay triangulation graph of the avatar positions and D^k, the k^{th} power graph of D. For simplicity we employ the same notation D^k for the corresponding adjacency relation. Thus, for a relation D^k the neighborhood of a vertex v is $D^k(v)$, the set of adjacent vertices. As a reminder, v and u are said to be neighbors in the power graph D^k if their distance in D *–i.e.*, the minimum number of hops between them– is at most k:

$$D^{k+1}(v) = D^k(v) \cup D^1(D^k(v))$$

The first three levels of this relation are represented in Figure 1a. To improve readability, the dual Voronoi tessellation is depicted, emphasizing each level.

The number of neighbors of an avatar v, *i.e.*, the size or cardinality of $D^k(v)$, noted $|D^k(v)|$, is not constant and depends on the distribution of other avatars in the proximity. To determine the range in which the number of neighbors evolves, we simulated avatar positions with distributions: uniform, Gaussian, and power law, and varying parameters. A power law avatar distribution is considered realistic (Legtchenko,

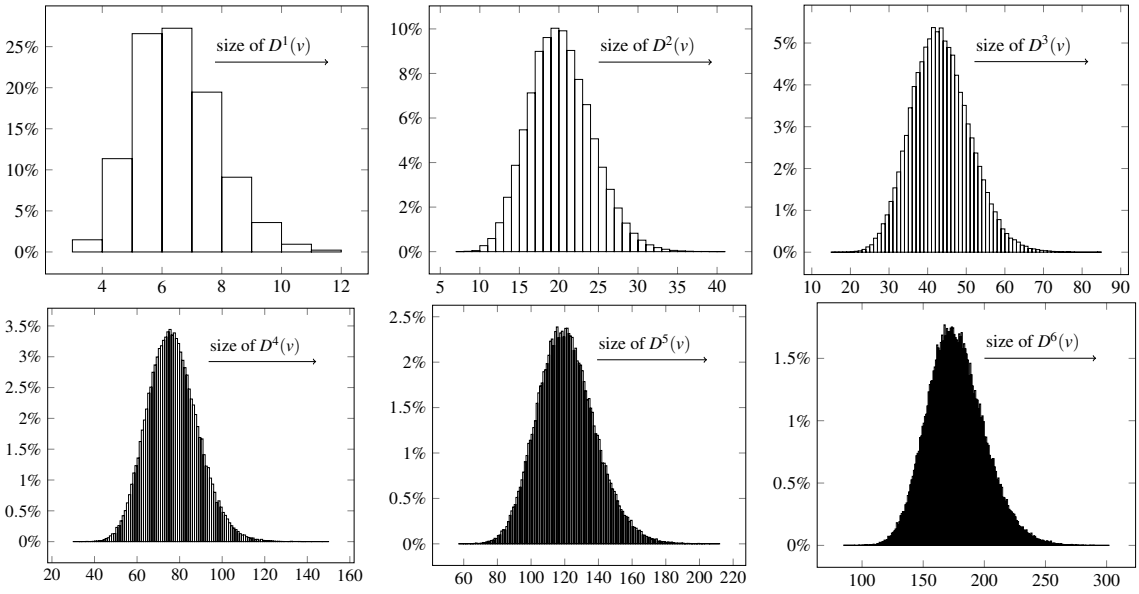


Figure 2: Distribution of the number of neighbors for D^k.

Monnet, and Thomas 2010), (Varvello et al. 2011), (Yahyavi and Kemme 2013). We summarize the outcome of the simulations for the power law distribution for D^1 to D^6 in Figure 2. We trace the percentage of avatars v with a given size of $D^k(v)$. Due to the local nature of the Delaunay triangulation, these percentages are stable, in the range of a Gaussian bell distribution, regardless of the total number of avatars. Actually, measurements with uniform and Gaussian densities produce very similar results. Only configurations built carefully and on purpose, with perfectly aligned positions, have been able to grow the neighborhood to large numbers.

Our notion of interest management based on Delaunay triangulations enjoys the following properties:

i) The number of neighbors falls within a range, providing the necessary setting to practical scalability, see Figure 2. This is a mandatory condition.
ii) The relation is symmetric, since D^k is an undirected graph. Bob sees Alice whenever she sees him.
iii) The nearest avatars are neighbors. In D^k the k-nearest avatars are guaranteed to be connected and in practice the number of nearest neighbors is greater (local properties of Delaunay triangulations).
iv) A similar property with three avatars: If they are alone inside a bubble they are all neighbors.
v) The number of meaningful neighbors to ensure social interaction varies depending on the intended application and features. We offer the possibility to customize it by using a suitable Delaunay power graph. For our current evaluation of Kiwano we have chosen $k = 3$ which yields a minimum of 15 neighbors and guarantees at least 30 neighbors for 96% of the avatars.

Condition (ii) avoids invisibility. It also ensures that interaction is perceived by all those involved. When an avatar sees another they are both able to interact. Bob sees Alice whenever she kisses him. Conditions (iii) and (v) circumvent the limitations imposed by a fluctuating avatar density. In an open space one sees where are the closest neighbors, while in a crowd one pays attention to those in the immediate proximity, avoiding the "empty world" effect.

We have seen in the previous section that the most common interest management technique, the set of avatars within a distance, does not satisfy these conditions because the density of avatars may vary sharply, if the area is adjusted for each avatar. The number of neighbors is bounded ((i), (iii), (v) are satisfied) but the relation is asymmetric ((ii), (iv)).

Providing a symmetric relation, the composition with another symmetric relation preserves this property. This is especially convenient when one needs to create the 'fog' of the virtual words, that is, to limit the visibility capacities to a fixed distance. For instance, perceiving the Delaunay neighbors in a fixed range distance is, indeed, a symmetrical relation.

3.3 Distributed Delaunay^K overlay

To make Kiwano scalable in the cloud, it is fundamental to distribute the computation of D^k and to update the structure dynamically. Here we explain how we construct and maintain a D^k graph, or DD^k, in a distributed manner among the nodes of the system, a system we call *the DD^k overlay.*

Each node i of the DD^k overlay *hosts* a set a of avatars: The node receives the position updates from these avatars and is responsible to notify them about their neighbors. Avatars are represented as vertices in the Delaunay triangulation at their respective positions. The set of vertices of the hosted avatars is denoted H_i or, when no confusion is possible, simply H. An avatar is hosted by only one node, therefore, the sets H_i are disjoint two-by-two.

We extend the definition of neighborhood for sets of vertices. For U, a subset of the vertices of the global graph D^k, the neighborhood of U is the union of all neighborhoods of its vertices:

$$D^k(U) = \bigcup_{u \in U} D^k(u)$$

We denote by *core* the set of vertices that have their neighborhood included in H.

$$Core_i^k = \{v \in H_i : D^k(u) \subseteq H_i\}$$

Correspondingly, In_i, the *in-border* set is constituted by the vertices that have neighbors outside of H_i:

$$In_i^k = H_i - Core_i^k$$

To supply to the hosted vertices all their neighbors we need to correctly compute the neighborhood relation for all vertices in H_i. More precisely, we must consider some of the vertices outside of H_i, namely, the neighbors of the vertices on the in-border. We name this set *out-border*:

$$Out_i^k = D^k(H_i) - H_i$$

The set of all vertices of node i, i.e., the *indexed vertices:*

$$V_i^k = D^k(H_i) \ \cup \ H_i = Core_i^k \ \uplus \ In_i^k \ \uplus \ Out_i^k$$

Each node i computes locally a Delaunay triangulation of its vertices V_i and the corresponding power (sub)graph D_i^k. We note $D_i^k|_{H_i}$ the D_i^k relation –and the derived graph– for the local set of hosted vertices H_i:

$$D_i^k|_{H_i} = \{(u,v) \in D_i^k : u \in H_i\}$$

These definitions enable us to show that the computation of the D^k neighborhood relation is distributable.

Theorem 1 (Distributed Delaunay^K graph) The global Delaunay triangulation graph D^k is the union of the locally computed $D_i^k|_{H_i}$:

$$D^k = D_1^k|_{H_1} \ \cup \ D_2^k|_{H_2} \ \ \cup \ \ldots \ \cup \ D_n^k|_{H_n}$$

Proof. *(Sketch)* Following the definition of the restricted neighborhood relation $D_i^k|_{H_i}$ every such relation i becomes thus asymmetric. For the ordered pairs (u,v), $u \in In_i$ if and only if $v \in Out_i$. However, for each node, all its in-border objects belong to at least one other node's out-border. In other words, $u \in In_i$ if and only if there is at least one node j such that $u \in Out_j$. □

Since Delaunay triangulations can be represented by their dual Voronoi tessellations, we define the *zone* of a node as the area composed by the Voronoi cells of its hosted vertices in H. Figure 1b pictures a zone and its out-border for $k = 3$. The $Core^3$ is white, the in-border is red and the out-border blue. We emphasise the neighborhood levels.

As the size of the neighborhood of each vertex is bounded, so the size of the out-border set is bounded. Also, as sets H_i are disjoint two-by-two, so are the relations $D_i^k|_{H_i}$. However, sets V_i overlap on their borders. When $V_i \cap V_j \neq \emptyset$, nodes i and j are *connected* in the overlay and are said to be *neighboring DD^k nodes.* Since the global D^k relation is symmetric, if $V_i \cap D^k(H_j)$ is not empty, then neither $V_j \cap D^k(H_i)$ is empty. In this case we say that D_i^k and D_j^k are neighbors. Correspondingly, the two DD^k nodes i and j are said to be neighbors. To summarise, *the neighborhood relation and the DD^k overlay form connected undirected graphs.*

4 ALGORITHMS

The chosen Delaunay data structure ensures locality: position updates affect (1) an avatar's old and new neighbors and (2) their respective DD^k nodes. In this section we explain how the data structure is maintained efficiently when positions are continuously updated.

4.1 A dynamic self-adaptive data structure

The *load* of the entire system depends on all the indexed vertices. As vertices on the borer are indexed on multiple servers, they consume resources (CPU and bandwidth). The aim is having most of the neighbors hosted by the same node, forming contiguous zones. In a highly dynamic environment each move tends to disrupt this property. In what follows we show how the distributed data structure is balanced among Kiwano nodes for efficient operations. This means minimizing the overlap at the borders and the number of neighboring nodes.

Avatar insertion. When an avatar joins the world, it needs to connect to a node in order to receive the up-to-date set of neighbors for its position. The role of the *dispatcher* is to select the entry node in the overlay (see Figure 3).

The dispatcher maintains a simplified map of the DD^k nodes and selects the right entry with high probability. Each node i of this DD^k map is represented by the barycenter of its hosted H_i vertices. The nodes update the map regularly regularly with the value of their barycenter. The dispatcher selects then the node whose barycenter is the nearest to the avatar's insert position and forwards the insert query to that node. This behaviour favorises a compact shape for zones.

The avatar sends its initial position to the dispatcher, which selects the best DD^k node to handle the avatar. The insertion query is then forwarded to the selected node that will add it to H, the set of *hosted* avatars. The node will, from now on, maintain the avatar's neighborhood and receive future position updates.

Transfer on position update. In a dynamic environment avatars move, frequently update their position and thus their neighbors change. When most of the neighborhood spans over another node, the vertex is *transferred* to that node. This behavior minimizes the size of the borders.

Load balancing. Avatar distribution can vary in time and some nodes may have more load than others. To avoid having most of the charge handled by just a few ones, nodes regularly send information about their load to their neighboring nodes. When the load of two neighboring nodes significantly differs, the most loaded one transfers to the other node avatar vertices from its in-border until they are balanced. In this way the vertices are step-by-step, peer-to-peer, partitioned into sets of similar sizes. The vertices to be transferred are selected to minimise the distance to the barycenter, again, favoring a compact shape for zones.

Node addition. When the total load of the system exceeds the resources available across the overlay, new nodes need to be added. When joining the system, a node must know at least one vertex and its

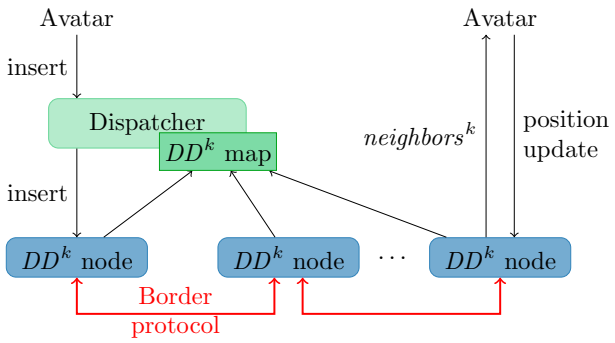


Figure 3: Distributed Delaunay^k overlay.

neighborhood to be able to maintain the borders with its neighboring nodes. In other words, the smallest zone is the Voronoi cell of a vertex. A new, empty node queries the dispatcher, which selects the node with the highest load. The selected node partitions the set of hosted vertices H_i in two disjoint subsets $U_1 \uplus U_2 = H_i$ and transfers one of them to the new node. They also communicate the outer borders to maintain the neighborhood in the node overlay. The two subsets U_1 and U_2 are selected using the barycenter criterion to produce two compact sets with small border overlap.

Reshaping. Even when the load is similar, the size of the borders may be too large leaving place to only a few internal vertices. For instance, if a node hosts one vertex, the size of the border is roughly 70. To avoid this, DD^k nodes regularly transfer to their neighboring nodes vertices on the in-border. The vertices to be transferred are selected to favorise compact zones.

4.2 Periodical Update

When avatars join, leave, and update their positions, DD^k nodes recompute the neighborhood, D_i^k to deliver, as soon as possible, updates to avatars. This is the *raison d'être* of the DD^k overlay. We identified two possible manners to compute the D_i^k: (1) **incrementally**, each position update is processed at a time and (2) **periodically**, incoming messages are aggregated and processed in batch. We initially implemented the incremental method that notifies clients of each update immediately. For one level of neighborhood, D_1^k, the system performs acceptably, but for two or more levels there are too many messages and overhead to maintain the borders. In Kiwano we implemented a periodical neighborhood computation. The Delaunay graph is recomputed ten times per second. Virtual worlds are implemented in a similar way, having a discrete loop in which events are periodically executed in batch with frequencies (or frame-rates (Bharambe, Douceur, Lorch, Moscibroda, Pang, Seshan, and Zhuang 2008), (Yahyavi and Kemme 2013)) of 10-20x/sec, e.g., MOVIES (Dittrich, Blunschi, and Salles 2011). Also, this delay does not apply to updates on known neighbors. Avatar data –including the position– can be notified directly, without passing through the DD^k overlay, see Section 5.

Avatar insertion. A node creates a new vertex when an avatar is (1) inserted by the dispatcher, (2) transferred from a neighboring node or (3) appears on the out-border. It is then queued to be inserted in the triangulation. When it is inserted or transferred, the avatar is also hosted –its vertex is added to H– and notified about the node that handles future position updates. Otherwise, the vertex is just added to the out-border and used to compute the D^k relation.

Avatar removal. When it leaves the world or has not updated its position for a long time, it is queued for deletion.

Position updates. For vertices in H updates come from the avatar. For the out-border thy come from a neighboring node. Updates are collected and the affected neighborhoods are notified after the subsequent D_i^k computation.

D_i^k computation. Before recomputing the D_i^k relation, the set V_i is updated with the pretreated events like border updates or avatar inserts, transfers, and removals. After the computation of the D_i^k relation, all hosted avatars receive notifications about the changes in their neighborhood. Of course, they will receive notifications only if there are any differences, *i.e.*, sets $D_i^k(u)$ - $oldD_i^k(u)$ or $oldD_i^k(u)$ - $D_i^k(u)$ are non empty. Also, neighboring nodes are informed and updated of all vertices that are in their out-borders. The cyclic computation of D_i^k is described in Algorithm 1.

Algorithm 1 Periodical update on each DD^k node

```
loop
  oldD_i^k ← D_i^k
  update V_i with the inserted, deleted, and updated vertices
  recompute D_i^k for the local set of vertices V_i
  for all u in H_i do
    inform u to add D_i^k(u) - oldD_i^k(u), if any
    inform u to discard oldD_i^k(u) - D_i^k(u), if any
  end for
  update and optimise the border with neighbour nodes
end loop
```

Summary. Vertices of the out-border have their positions updated by the node that hosts them. So, updates for the vertices on the border are forwarded to the concerned neighboring nodes. If a node receives a border update for a vertex not already in its out-border, the vertex is added to the set *Out*. This node to node communication ensures a coherent Distributed Delaunay computation in a decentralized system.

The amount of exchanged border messages grows with the size of the borders and the number neighboring nodes. Therefore, we need to minimise the number of vertices on the borders and the number of neighboring nodes. The reshaping process and, in general, the way the avatars are transferred from one node to another are aimed at optimizing these values using heuristics. At regular time intervals, the load balancing and node addition processes distribute the load evenly among all nodes. However, every avatar movement tends to break the equilibrium. For some frequency of updates or amplitude of movements the system will witness a degradation in performance. The performance of our actual implementation is described in Sec. 6.

5 SYSTEM ARCHITECTURE

In this section we describe the *cloud architecture* that supports the discussed algorithms and exposes the functionality with a *public API*. By simply connecting to this API we open the possibility for *virtual world interoperability*.

Transparency. The DD^k nodes will not be exposed directly through the API. This would lead to too frequent reconnections from the clients. We introduced *proxy* nodes, each avatar having a proxy as single entry point for the entire session, as depicted in Figure 4. Proxies forward messages between clients and DD^k nodes hiding the distribution of the Kiwano algorithm from the developers. Their charge depends linearly on the number of avatars hosted and do not pose a scalability problem. All proxy nodes report on their availability to an *allocator* which assigns proxies to the newcomers such that the load among the proxies is balanced.

Public API and Interoperability. We envisaged Kiwano to be deployed in the cloud and to be accessible via an API to virtual world developers. The exposed functionality allows an avatar to: (1) update position and state, (2) gracefully leave the virtual world, (3) get all neighbors, (4) send message to all/one neighbor(s). Kiwano responds with the following messages: (1) full list of neighbors and (2) notification message or neighbor position. When connecting multiple virtual worlds to Kiwano, avatars indexed together, thus the interoperability. Our examples include HybridEarth and a Minecraft plugin (Valero, Diaconu, and Keller 2013), (de Campredon, Diaconu, Keller, and Triponez 2014), (Diaconu 2015).

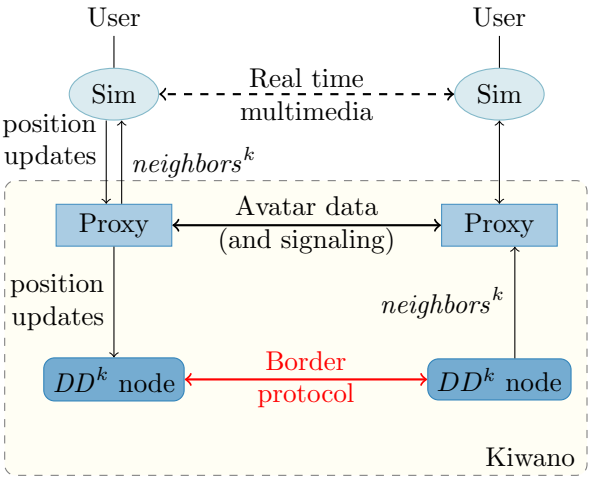


Figure 4: Kiwano distributed architecture.

6 EVALUATION

Kiwano is implemented in python using bindings to C++ Delaunay triangulations on a sphere (Caroli, de Castro, Loriot, Rouiller, Teillaud, and Wormser 2010).

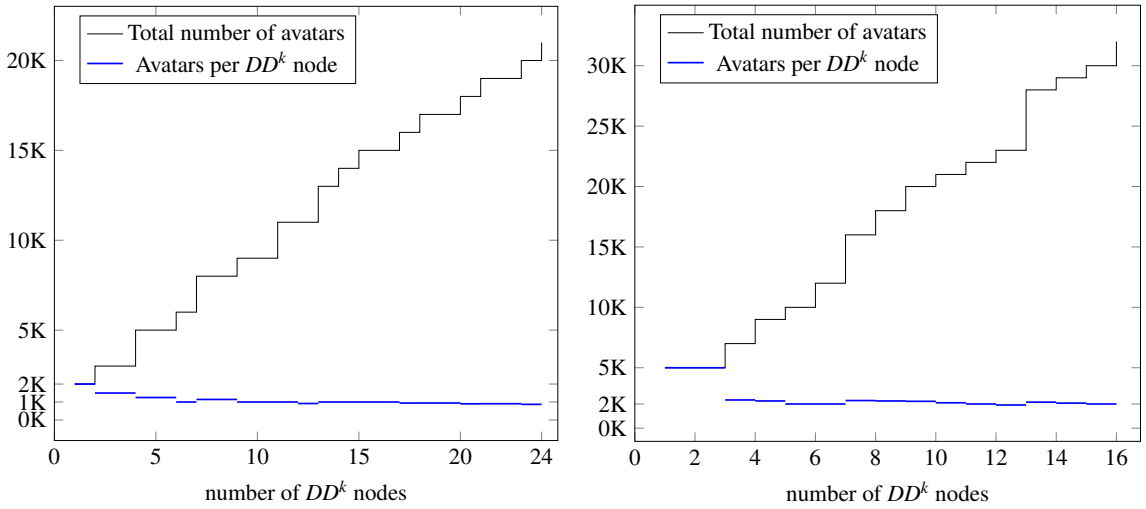
Settings. To perform the evaluation, we employed a heterogeneous system. We disposed of virtual and physical machines of diverse configurations and performances located in four different physical places. DD^k nodes, the dispatcher, and the allocator ran on four 8-core Intel i7 CPU 920 at 2.67GHz, 64bit, with 24GB of RAM located in the same data center. Each DD^k node was running on a different core. All nodes were communicating via ethernet regardless wether they were running on the same station or remotely. For running the proxies we had at our disposal:

- 15 physical CPUs in a data center, Intel Xeon at 2.00GHz, 64bit, 2GB of RAM,
- 30 virtual machines in a different data center, QEMU Virtual CPU, 64bit, 1.5GB of RAM,
- 6 desktop computers in our office, various CPUs at 32bit, all with 4GB of RAM.

Measurements. The avatar simulators ran on proxies and followed a large scale power-law distribution inspired from Blue Banana (Legtchenko, Monnet, and Thomas 2010), (Diaconu 2015). The proxies reported to the allocator their state, load, and throughput of incoming and outgoing messages. The DD^k nodes reported to a monitoring server every 10 seconds average values for message delays, number of vertices, size of borders, exchanged messages, etc.

Initially, we ran the monitoring server, the dispatcher, the allocator and several idle DD^k nodes. New DD^k nodes can be added to the system whenever they are needed. The dispatcher was configured to use an idle DD^k node once the average delay between two neighborhood notifications decreased under a fixed threshold. We launched avatars, one thousand at a time and waited several seconds in between for the system to stabilise. During this time, the simulated avatars sent their updates with the programmed frequency. Once the system was stable (i.e., all insertions from the dispatcher were complete and the measured parameters became constant) the reports were recorded by the monitoring server for all nodes in the system.

Scenarios. The required frequency for updates in virtual worlds can vary greatly. Second Life and MMORPGs (World of Warcraft) are centered around social interaction, avatar customization and the neighborhood does not change frequently (Yahyavi and Kemme 2013). Occasional neighborhood updates can be delayed up to a few seconds. Temporal resolution for FPS games needs to be as high as 10-30 frames/sec (Yahyavi and Kemme 2013). In addition, we evaluated neighborhood changes using a real



(a) Action game scenario

(b) Social virtual world

Figure 5: Kiwano Scalability Evaluation.

dataset which captures several hours of soccer player movements (Pettersen, Johansen, Johansen, Berg-Johansen, Gaddam, Mortensen, Langseth, Griwodz, Stensland, and Halvorsen 2014) where positions are reported every 100ms. However, most of these movements do not impact the Delaunay graph and, hence, avatar neighborhood remains unchanged. As a result we evaluated two different scenarios that match these demanding requirements:

Action games: Position updates are frequent, every 0.5 seconds. The delay between two neighborhood updates does not exceed 0.2 seconds. If the delay exceeds this value, a DD^k node is added to the overlay.

Social virtual worlds: Each simulated avatar updates their position every 4 seconds. DD^k nodes send neighborhood updates for each hosted avatar. In this scenario the bottleneck becomes the number of incoming messages, therefore the addition of DD^k nodes is triggered when messages to the proxies are lost.

6.1 Results

Our hardware settings allowed us to perform the evaluations with 22.000 moving avatars hosted by 24 nodes for action games scenario (Figure 5a) and 32.000 avatars hosted by 16 servers for the social virtual world scenario (Figure 5b). This is orders of magnitude beyond current state-of-the-art. Most importantly these evaluations show that sub-quadratic scalability is feasible. In our range of evaluation, we observe, as expected, that the number of avatars per node tends to stabilise around 1000 avatars per node for the action game scenario and 2000 avatars per node for the social world scenario. Meaning that, from a certain point, scalability is linear, or, the measured cost per avatar is, as theoretically predicted and expected, constant.

Intuitively, we want to minimise the number of avatars on the borders because their position updates need to be propagated to the neighboring nodes. Basically, avatars on the borders are represented on two or more nodes. Results in Figure 5 support this intuition. In both scenarios, when all only one node is needed, it can host twice as many avatars per node then in the case when 10 or more nodes are are employed. Filtering mechanisms may also be used to increase the number of avatars per DD^k node by lowering the average frequency of updates. When positions change only slightly, oftentimes they do not trigger changes

in the neighborhood and they can be sent directly to the neighbors, without recomputing the Delaunay triangulation.

Some issues may arise. When the number of nodes grows, the probability to go across several nodes increases. For example, if we have two DD^k nodes, a teleportation, in the worst case, would transfer the vertex to the other node. But with more nodes, the vertex can be transferred subsequently, from node to node multiple times before reaching the correct node.

7 CONCLUSION

We described Kiwano, a solution for scalable virtual worlds without borders. We handle separately the main source of load: avatar mobility. Kiwano employs a distributed Delaunay triangulation to provide each avatar with a constant number of neighbors independently of their density or distribution. The avatar-to-avatar interactions and related computations are bounded, allowing the system to scale. We evaluated our implementation with tens of thousands of avatars connecting to a Kiwano instance running in the cloud, across several data centers, outperforming by many orders of magnitude actual solutions.

Kiwano is a valid solution for real world projects. Our intention is to provide the first massively distributed and self-adaptive solutions for virtual worlds suitable to run in the cloud. Kiwano API and docs can be accessed at `http://kiwano.li` and virtual worlds that use it and interoperate at `http://hybridearth.net` and `http://manycraft.net`.

REFERENCES

Almroth, D. 2010. "Pikko Server". In *Erlang User Conference*. Stockholm, Sweden.

Bharambe, A. R., J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. 2008. "Donnybrook: Enabling Large-scale, High-speed, Peer-to-peer Games". In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 389–400. Seattle, WA, USA.

Buyukkaya, E., M. Abdallah, and R. Cavagna. 2009, Jan. "VoroGame: A Hybrid P2P Architecture for Massively Multiplayer Games". In *IEEE Consumer Communications and Networking Conference (CCNC)*, 1–5. Las Vegas, NV, USA.

Caroli, M., P. M. M. de Castro, S. Loriot, O. Rouiller, M. Teillaud, and C. Wormser. 2010. "Robust and Efficient Delaunay Triangulations of Points on Or Close to a Sphere". In *9th International Symposium on Experimental Algorithms, (SEA)*, 462–473. Ischia Island, Naples, Italy.

de Campredon, J., R. Diaconu, J. Keller, and E. Triponez. 2014. "HybridEarth: Social Mixed Reality at Planet Scale". In *IEEE Consumer Communications and Networking Conference (CCNC) (CCNC) - Demos*. Las Vegas, NV, USA.

de Oliveira, J. C., and N. D. Georganas. 2003. "VELVET: An Adaptive Hybrid Architecture for VEry Large Virtual EnvironmenTs". *Presence* 12 (6): 555–580.

Debeauvais, T., A. Valadares, and C. V. Lopes. 2012. "Evolution of scalability with synchronized state in virtual environments". In *IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE)*, 142–147. Munich, Germany.

Diaconu, R. 2015. *Scalability for Virtual Worlds*. Ph. D. thesis, Université Pierre et Marie Curie, Paris VI.

Diaconu, R., and J. Keller. 2014. "OneSim: Scaling Second Life with Kiwano". In *Proceedings of International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 8:1–8:2. Singapore.

Diaconu, R., J. Keller, and M. Valero. 2013. "Manycraft: Scaling Minecraft to Millions". In *Network and System Support for Games (NetGames)*, 1:1–1:6. Denver, CO, USA.

Dittrich, J., L. Blunschi, and M. A. V. Salles. 2011. "MOVIES: Indexing Moving Objects by Shooting Index Images". *GeoInformatica* 15 (4): 727–767.

Ellis Hamburger 2013, July. "Largest space battle in history claims 2,900 ships, untold virtual lives". http://www.theverge.com.

Hu, S.-Y., J.-F. Chen, and T.-H. Chen. 2006, July. "VON: A Scalable Peer-to-Peer Network for Virtual Environments". *IEEE Network Journal* 20 (4): 22–31.

Keller, J., and G. Simon. 2002. "Toward a Peer-to-Peer Shared Virtual Reality". In *International Conference on Distributed Computing Systems (ICDCS)*, 695–700. Vienna, Austria.

Knutsson, B., H. Lu, W. Xu, and B. Hopkins. 2004. "P2p Support for Massively Multiplayer Games". In *IEEE Computer and Communications Societies (INFOCOM)*. Hong Kong, PR China.

Lake, D., M. Bowman, and H. Liu. 2010. "Distributed Scene Graph to Enable Thousands of Interacting Users in a Virtual Environment". In *Network and System Support for Games (NetGames)*, 1–6. Taipei, Taiwan.

Legtchenko, S., S. Monnet, and G. Thomas. 2010. "Blue Banana: resilience to avatar mobility in distributed MMOGs". In *IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, 171–180. Chicago, IL, USA.

Pettersen, S. A., D. Johansen, H. Johansen, V. Berg-Johansen, V. R. Gaddam, A. Mortensen, R. Langseth, C. Griwodz, H. K. Stensland, and P. Halvorsen. 2014. "Soccer Video and Player Position Dataset". In *ACM Multimedia Systems Conference (MMSys)*, 18–23. New York, NY, USA.

Valero, M., R. Diaconu, and J. Keller. 2013. "Manycraft: Massively Distributed Minecraft". In *Network and System Support for Games (NetGames Demo)*, 17:1–17:3. Denver, CO, USA.

Varvello et al., M. 2011. "Exploring Second Life". *IEEE/ACM Transactions on Networking* 19 (1): 80–91.

Yahyavi, A., and B. Kemme. 2013. "Peer-to-Peer Architectures for Massively Multiplayer Online Games: A Survey". *ACM Computing Surveys* 46 (1): 9.

Yu, A., and S. T. Vuong. 2005. "MOPAR: A Mobile Peer-to-peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games". In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 99–104. Stevenson, WA, USA.

AUTHOR BIOGRAPHIES

RALUCA DIACONU is a Research Associate at the Computer Laboratory at University of Cambridge, UK, working in the area Internet of Things and Distributed Computing. Dr Diaconu holds a PhD in Distributed Computing at University Pierre and Marie Curie, Sorbonne Universités in Paris, France, for her work in distributed algorithms and architectures for scaling up hybrid and virtual worlds. Her email address is raluca.diaconu@cl.cam.ac.uk.

JOAQUÍN KELLER graduated in Mathematical Logic at Université Denis Diderot and obtained a PhD in Distributed Systems from Université de Versailles. He has been leading research in virtual worlds for more than ten years. He is the designer of Solipsis "the first peer-to-peer virtual world" and cofounder of Twinverse, a technology company for social location based services and virtual worlds. He is now a senior researcher at Orange Labs where he has launched HybridEarth, a mixed reality world spanning the planet, and Kiwano, a scalable distributed infrastructure aimed at allowing an unlimited number (i.e., millions and more) of users to simultaneously interact in a virtual environment. His email address is joaquin.keller@gmail.com.