Multimed Tools Appl (2009) 45:83–107
DOI 10.1007/s11042-009-0287-7

The partial migration of game state and dynamic server selection to reduce latency

Paul B. Beskow · Knut-Helge Vik · Pål Halvorsen · Carsten Griwodz

Published online: 5 May 2009

Abstract Massively multi-player online games (MMOGs) have stringent latency requirements and must support large numbers of concurrent players. To handle these conflicting requirements, it is common to divide the virtual environment into virtual regions. As MMOGs are world-spanning games, it is plausible to disperse these regions on geographically distributed servers. Core selection can then be applied to locate an optimal server for placing a region, based on player latencies. Functionality for migrating objects supports this objective, with a distributed name server ensuring that references to the moved objects are maintained. As a result we anticipate a decrease in the aggregate latency for the affected players. The core selection relies on a set of servers and measurements of the interacting players latencies. Measuring these latencies by actively probing the network is not scalable for a large number of players. We therefore explore the use of latency estimation techniques to gather this information.

Keywords Massively multi-player online games · Latency · Estimation · Server architecture

P. B. Beskow (✉) · K.-H. Vik · P. Halvorsen · C. Griwodz
Simula Research Laboratory, Martin Linges v 17, 1364 Snarøya, Norway
e-mail: paulbb@ifi.uio.no

P. B. Beskow · K.-H. Vik · P. Halvorsen · C. Griwodz
Department of Informatics, University of Oslo, Gaustadalléen 23, 0373 Oslo, Norway

K.-H. Vik
e-mail: knuthelv@ifi.uio.no

P. Halvorsen
e-mail: paalh@ifi.uio.no

C. Griwodz
e-mail: griff@ifi.uio.no

1 Introduction

"Lagger!" is a likely expression to hear uttered in a real-time interactive online game. This term addresses players with excessive latency, which in the gaming community is colloquially referred to as *lag* (no positive connotation is implied by this term). A player with high latency will inadvertently have a negative effect on the perceived quality of the game play [4, 14]. This occurs, as most online games are based on a client-server model, where events are collected at the server, and distributed to the interacting players. By its nature, if one player's connection is comparatively slower, any added delay is not isolated to the player alone. It will propagate to the interacting parties and potentially result in inconsistencies, which the server must then recover from. While having minor effects on the outcome of the game, it results in a perceived deterioration to the quality of interaction [19]. As such, *low latency* for *all players* is a prevalent goal. In this paper, we extend our body of work on migration and server selection [7] and expand the latter to include an exploration of latency estimation techniques to obtain the required network information in a scalable way. Thus, we use migration techniques to move the processing to a more appropriately placed node found using a core selection algorithm based on obtained network information.

The latency requirements of games vary greatly, ranging from 100 to 1000 ms [16]. One factor affecting the latency seen by the players is their physical distance to the server. As such, to achieve a satisfactory quality of interaction, the player may need to be located within a reasonable proximity to the game server, or that we have a dynamic selection of servers; to be in the proximity of most of the users.

As an example, consider world-spanning MMOGs, which are persistent online worlds that allow thousands of players to interact concurrently in a virtual environment. To support this many concurrently interacting players, the virtual environment is commonly split into virtual regions. This makes it possible to distribute the regions across a number of (possibly geographically distributed) servers. As the regions are logically decomposed, each server is responsible for handling some regions and the players interacting in each region. Since a player's proximity to the server impacts the latency, and in turn, latency is an integral factor for the playability of an online game, this raises an interesting question: *How can we optimize the placement of a virtual region given the interacting players and a set of globally distributed servers?*

Core selection provides a solution to this server selection problem. Given a set of players and servers (and proxies), it finds an appropriate server (or proxy) for placing a virtual region. It measures the latency of each player to each server and locates the server that provides the minimum diameter (lowest of the highest pair-wise latencies) in order to take into account that a game event sent from one player must reach all other players within the latency constraints.

Core selection depends on latency measurements from the servers to the players. These latencies may be obtained by actively probing and monitoring the network, but this is not scalable due to the potentially large number of players and servers/proxies, i.e, n^2 measurements in the worst case. As such, we investigate the use of latency estimation techniques for use with core selection, focusing on Vivaldi [18] and Netvigator [41]. Estimation techniques measure a sub-set of the links, and then estimate the remaining links based on these measurements. We consider the impact

of estimation on the quality of the core selection process, because although these techniques are scalable, a penalty arises from their estimation accuracy. Results show that Netvigator yields accurate latency estimates, while Vivaldi is more inaccurate, but still usable. Netvigator is harder to set up than Vivaldi, but they are both likely candidates for use in distributed interactive applications.

Once a server has been selected, we use our migration functionality to move the active region to its new location. To maintain an optimized set of references to the migrated game state, we use a distributed name service. On the background of existing work in distributed systems, we believe that the combination of *core selection, latency estimation, a distributed name server and migration*, a viable solution for creating a globally distributed game, which is capable of *lowering the overall latency of the interacting players.*

The rest of the paper is organized as follows: in Section 2, we look at the basis for our assumptions, and take a closer look at some related work. In Section 3, we look at the migration functionality and how it is supported by the distributed name service. In Section 4, the core selection process is described in detail, and how it can be applied to the scenario we have described. In Section 5, we describe different classes of latency estimation techniques. In Section 6, we evaluate our migration functionality and the core selection process (using both measured and estimated input). In Section 7, we discuss major aspects of this work. Finally, we summarize our findings in Section 8.

2 Background and related work

2.1 Game characteristics

The body of work that analyzes game traffic has grown considerably in the recent past. The main conclusions in our scenario are that 1) game traffic varies strongly with time and the attractiveness of the individual game [12, 25], 2) some latency is tolerable [14] as long as it does not exceed the threshold for playability from 100 ms to 1000 ms depending on the type of game [16] and 3) geographical dispersion of players in an online game depends heavily on the time of day [24]. In addition to these works, we have analyzed packet traces (see [29]) from Funcom's popular role-playing MMOG *Anarchy Online* [26]. Statistics from the traces reveal that there lies a potential latency improvement in using our migration middleware. For example, in one of the game regions, with measurements recorded within a time span of about one hour, we found approximately 175 distinct connections. These are sorted according to their measured round-trip times (RTTs) in Fig. 1. With the knowledge that the servers are located in the US, the observed minimum latencies in the figures indicate that there are players concurrently located in the US, Europe and Asia. The number of players in different areas of the world also typically vary according to the time of day, and finding an appropriate location for the server might be of vital importance in order to meet the latency requirements. In Fig. 1 we can see that the majority of users are in the second range (20–140), as such, the average latency could be reduced by moving the analyzed game region to a server in Europe.

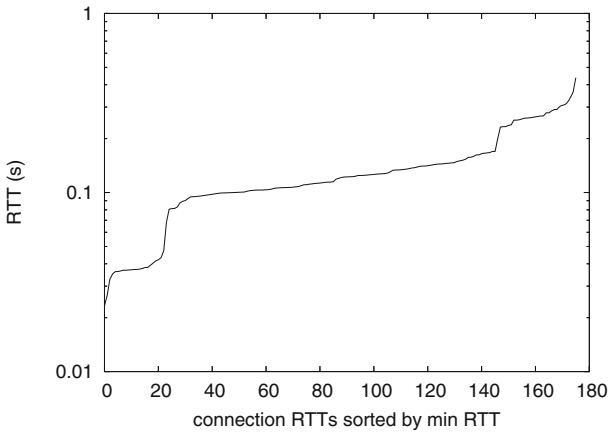


Fig. 1 Anarchy Online: connection RTTs sorted by min RTT

2.2 Migration techniques

Migration provides the functionality to move entities between servers/proxies/clients in a distributed system. These entities can take different forms and be of greatly varying granularity. In [13] Clark et al. migrate a live virtual machine from one node in a cluster to another, with as little as 60 ms of downtime. In a similar study, Nelson et al. [36] describe how a virtual machine can be transparently migrated with minimal impact on the user. At a lower granularity level we find DEMOS/MP [39], Sprite [33] and Mosix [3], which are all *NIX based operating systems that allow for processes to be migrated. They are required to capture all of the data and state associated with the process (the program data, its stack and registers: program counter, stack pointer, and so on) to facilitate the migration. JavaScript and SQL [22] are examples of systems that make code migration possible. JavaScript is client-side executed code, while SQL is primarily executed by the database management system at the server hosting the database system. Finally, Emerald [30] and Chorus/COOL [8] are examples of systems that make it possible to migrate objects during run-time execution of a program (per the object-oriented paradigm). Migration can be used to serve several purposes, such as dynamic load distribution, fault resilience, increasing resource locality or to facilitate system administration. In our case, we wish to achieve a combination of load distribution and increased resource locality by migrating game state to an optimal location (relative to the interacting users).

2.3 Server selection

Game server selection is an important facet of the playability for several types of online games. This is particularly true for games that are highly sensitive to latency, such as first person shooter (FPS) games. As such, the player's selection process is commonly guided by measuring dimensions that affect playability, such as latency and packet loss. This sensitivity to latency is commonly alleviated by having a high distribution, and availability, of servers. With respect to this, Chambers et al. [11] have looked at how server selection can be optimized for a single client, when given a set of available servers. In a further study, Claypool [15] notes that we regularly

find groups of players that wish to play together on a server, such as friends or clans (organized players). As such, he has investigated how server selection can be optimized from the perspective of a group of players. In two related studies by Armitage [1, 2], efficient ways of ranking servers in the discovery process itself are examined. These papers have in common that they consider server selection from the perspective of the player(s), and additionally assume a certain availability of servers (it is common for geographically coupled players, such as real life friends, to play against each other). For a world spanning game, however, where all users interact in the same game instance, such as an MMOG, there is often a limited number of servers to select from. In [34], Lee et al. present their heuristic for selecting a minimum number of servers satisfying given delay constraints (from the perspective of large scale interactive online games, such as MMOGs). Their aim, however, is to have well provisioned network paths in a centralized architecture. Thus, they do not consider the aspect of geographical dispersion of players. In a similar study, Brun et al. [10] investigate how a server's location can influence the fairness of a game, and how selecting an appropriate server impacts this fairness. They use an objective function, which they call *critical response time* to rank the servers.

2.4 Network estimation

It is straightforward to obtain a link's RTT in the Internet using measurement tools like ping and traceroute (or mechanisms like packet pair and packet train [20]). The drawback with ping is that it does not return any measurements if the target host is unreachable. Traceroute, on the other hand, returns latency measurements for each (answering) hop on the route. This may be valuable for some latency estimation techniques if they control network routers, and if the end-to-end reachability is limited. Furthermore, in the scenario of large-scale distributed interactive applications, it is currently not scalable to use measurement tools like ping or traceroute to actively monitor networks for their link latencies. Instead, latency estimation techniques that reduce the probing overhead should be applied.

Achieving full up-to-date knowledge of the network requires live latency monitoring and is not scalable for a large number of players and servers/proxies. In the best case $M \times N$ (for M servers/proxies, and N players) measurements are required, in the worst case N^2 (in the case where all nodes can be used as servers, i.e., a peer-to-peer setting) are required. This scalability problem is addressed by techniques that estimate link latencies, but the trade-off is their accuracy. In general, a latency estimation technique probes a (low) number of links to sample their link latencies, and then attempts to estimate the remaining links based on these probes. There are a multitude of latency estimation techniques. Many of the latency estimation techniques are likely to be usable in a distributed interactive application setting. However, the main comparative metric is whether or not the estimations are accurate enough.

The estimation techniques may be classified into one of three classes [23].

Landmarks-based latency estimation technique assign each node a point in a metric space, and aim to predict the latency between any two nodes. They use landmark nodes, a set of nodes that are used by others as measurement references for their relative position in the network. Examples of landmarks-based techniques

are Netvigator [41], NetForecast [23], Global Network Positioning (GNP) [37] and Practical Internet Coordinates [17].

Multidimensional-scaling based latency estimation techniques use statistical techniques for exploring similarities and dissimilarities in data. For example, a matrix of item-item similarities is used to assign a location for each item in a low-dimensional space [17]. Vivaldi [18] is such a technique.

Finally, *distributed network latency database* techniques use active measurements to build a knowledge base about the underlying network. These approaches have been designed to efficiently answer queries of the form: Who is the closest neighbor to node A in the network? Since these schemes are based on direct measurements they have better accuracy. They also inject more traffic into the network compared to the landmark-based and multidimensional-scaling based techniques. Meridian [45] is a technique that uses a distributed network latency database.

Distributed network latency database techniques are not desirable for our target area, because they are not designed to retrieve all-to-all link latencies. Instead, however, we focus on landmarks-based and multidimensional-scaling based latency estimation techniques. They are desirable for leader election scenarios, and discovering the closest neighbor for a node.

2.5 Summary

Our analysis of the Anarchy Online game traffic shows that players connect from all around the world (see Fig. 1). An approach to reducing the latency, both due to RTT and loss, is to dynamically find the center of the group of players and migrate the game objects to a server whose location is closer to the majority of the users. We can accomplish this reduction through core selection, which helps us determine which node to migrate the players to. An issue with core selection is that it depends on latency measurements to run correctly. Actively measuring these latencies with ping or traceroute is costly, and depending on the number of players, not scalable either. As such, latency estimation techniques offer a viable alternative to gather such information.

3 Moving worlds with migration

Core selection is applied to determine whether the current server hosting a virtual region (based on its player population) is optimal. In the case where it is able to locate a more appropriate server based on the latency to the active users, the MMOG moves the game state of that region to its new location. To accommodate this process, we have developed a middleware that is capable of performing such migration of game state, which consists of a number of interacting objects (following the object-oriented paradigm). Before migrating an object, we must know that all references to that object are maintained. To accomplish this, we use a name service. This service is responsible for keeping an up-to-date index for the location of objects in the distributed system, and redirect method invocations accordingly. As such, the reduction in aggregate latency is accomplished by decreasing the average response time of remote method invocations (RMI). The response time is decreased because we move objects closer to the majority of the players. To accommodate the

development cycle we have also written a tool that generates code to ease integration of the application with the middleware.

3.1 Name service

A name service maintains references to objects in the distributed system. This task can be accomplished in several ways, and Znati et al. [47] analyzed three approaches to implementing a name service; in the form of centralized, hybrid and distributed versions, which we have discussed thoroughly (in the context of MMOGs) in [5]. The conclusion is that the centralized model achieves acceptable performance only as long as the ratio of remote to local requests is kept reasonable. The performance of the hybrid model depends highly on the efficiency of the cache design, and with all other network conditions equal the relative response times of the distributed architecture were smaller. A distributed name service best suits our needs, primarily because efficiency is more important then consistency in this scenario. In addition, the following characteristics also have an impact on this decision:

1. There are thousands of concurrently interacting players.
2. The virtual environment is divided into virtual regions.
3. Players are dispersed physically as well as virtually.
4. The servers in the system are geographically distributed.
5. Code is shared so only data is migrated.
6. There occurs frequent object creation and destruction.
7. Efficiency is more important then consistency.

A distributed name service is more efficient then a centralized or hybrid approach because there is no overhead in binding an object with the name service. This, because each node has its own name service, to which objects are bound initially. Communication between nodes (and thus the name services) only becomes necessary when an object is migrated. Given that the servers in the system are geographically distributed, binding objects locally becomes quite beneficial. Other advantages are that there is no single point of failure, which implies that large parts of the application can continue running if a server were to fail. Looking up objects is also efficient, as we can directly query the node our name service has registered as the current care taker of the object. Though it is worth noting that this access time will depend on the number of times an object has been migrated (after its point of creation) and at which point in this chain the invocation is performed. For further details and a thorough discussion about the implementation of the name service, distributed references and migration see [6]. In the following section, we will solidify our understanding of the described mechanisms by looking at example of these concepts in use.

3.2 In action

Consider a system consisting of two servers, as seen in Fig. 2. The majority of the interacting players are European, with a couple of players connected from *Nuuk* and *Seattle*. Currently, all of the virtual regions are hosted in *Oslo*. Looking at Fig. 3a, we see that at least two players have been bound to their local name service and received an identifier (we do not show the other players in this example). Additionally, the *Nuuk* player references the *Seattle* player. This reference is not to the player object

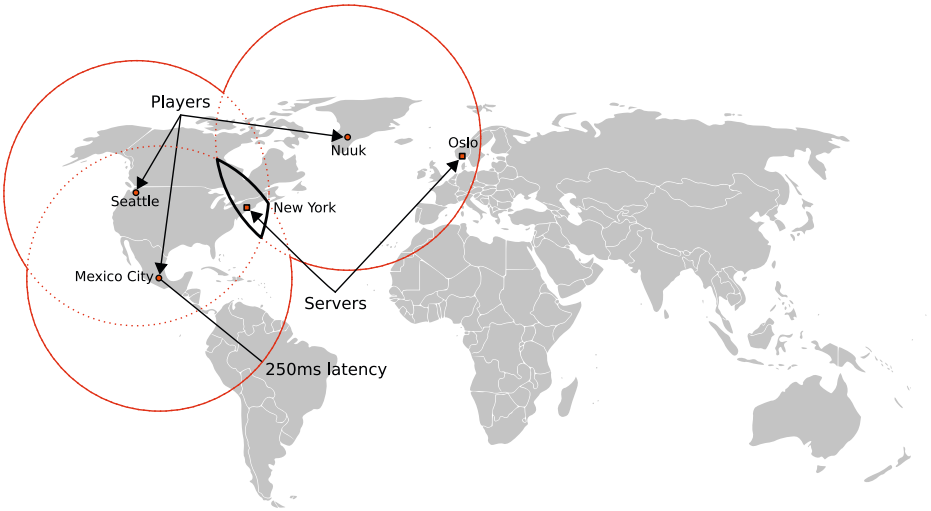
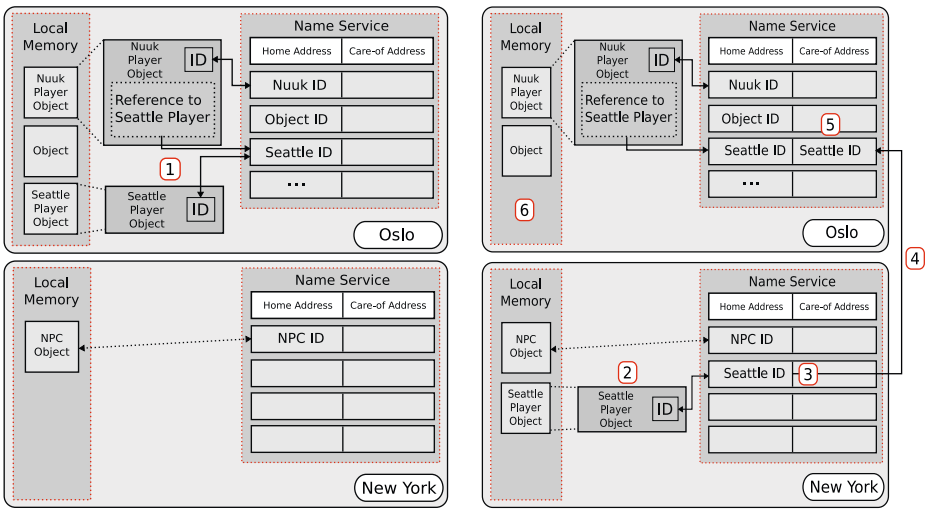


Fig. 2 Latency as physical distance

in local memory, as we might expect, but to the identifier (created earlier) in the name service. After some time the population density shifts towards an American dominance, triggering a core selection for each virtual region (we describe this in further detail in Section 4). For the region with our two players, the server in *New York* is deemed a better fit. A migration is triggered, with our two players migrated to the server in *New York*. The steps outlined in Figs. 3a (before) and 3b (during/after) guide us through this process.



(a) *Before* migration. (b) *After* migration.

Fig. 3 Server configuration (**a**, **b**)

First (1) we serialize the player object, which means a binary representation of the object is created. The data representing the serialized object is then transmitted to the receiving node, in this case the server in New York. The *second* step (2) consists of deserializing the object at the receiving side. This recreation is accomplished by initializing an object with the binary stream. Once the object has been recreated, it is bound to the local name service, which is the *third* step (3). At this point in time no reference between the original node and the new node exists. As such, the original node will have no way of forwarding requests. To resolve this problem, the new identifier assigned to the player object by the server in New York is returned to the original server in Oslo, which concludes the *fourth* step (4). After Oslo has updated its name service by associating the received identifier with the player object, which happens in step *five* (5), we can without worry remove the object from local memory; as is done in step *six* (6).

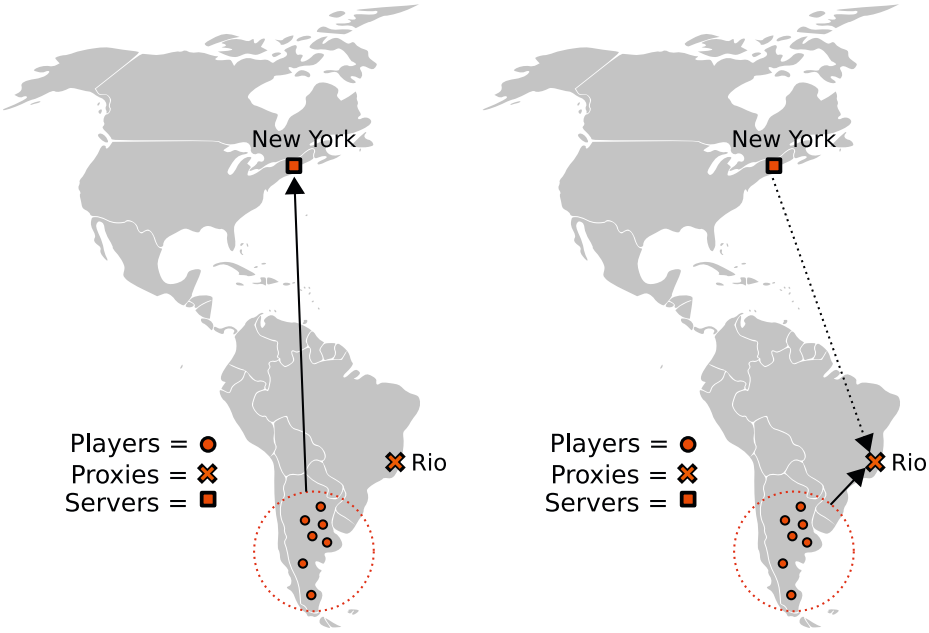
3.3 Code generation

Integrating code with a middleware, such as the one we have developed for migrating objects, can be tedious and error-prone work. To accommodate the development cycle we have written a tool for automatically generating skeletons that integrate with the middleware. The code generation tool is implemented as a Python script, which takes annotated C++ header files as input. To parse the C++ header files, we make use of the GCC-XML [32] parser, which is a tool that extends the open source GCC compiler, using its internal representation to produce XML output. Based on information obtained by processing the XML-file, we generate the required code to integrate the user-defined class seamlessly with the middleware. As mentioned, the skeleton generator expects a C++ class declaration as its input. In addition to normal C++ class syntax, GCC-XML allows for defining additional attributes. We make use of this ability to extend the C++ syntax with our own keywords. When an object is migrated, one does not necessarily want all the data to be serialized. We therefore provide a special keyword (*_serialize*) to specify the data to be serialized. Other suitable keywords will be introduced to support remote method invocation, mark the classes that can be migrated and so forth.

4 Core selection

Today's typical client/server model makes it easy to manage the global game state, but it has drawbacks. The server is a potential bottleneck, both in terms of computing and bandwidth capacity, and the latency depends heavily on the physical distance from each individual player to the server. In Fig. 4a, we illustrate an example where a centralized server stores the game state and thus cannot take into account the physical location of the players.

Proxy technology is a distributed option. In this model, we have an infrastructure with a centralized server and a set of distributed proxy servers. The proxies are used to increase the physical distribution of servers, where we aim to achieve much the same as content distribution networks (CDNs), i.e., have a distribution point in close proximity to the players for faster distribution of data. In this scenario, an efficient way to reduce latency can be to migrate game state to an appropriate server, close



(a) Centralized architecture (b) Proxy architecture

Fig. 4 Architecture (**a**, **b**)

to the center of a given group of players. Consider an MMOG with geographically distributed servers (see Fig. 2). At any time, a server can be hosting none, some or all of the virtual regions making up a virtual environment of a game instance. Furthermore, looking at Fig. 2, we can see how latency (reasonably) compares to a measurement of physical distance. In case of a 250 ms delay requirement from the server (e.g, according to the 500 ms pairwise latency requirement in RTS [14]), the bordered intersection indicates an optimal area for the server to be located. In Fig. 4b, the central server has migrated the game state to a proxy that is closer to the group of players.

Proxy technology allows a trade-off between client/server and peer-to-peer advantages and disadvantages. A pure peer-to-peer architecture then implies that the game state is distributed among the players, with no central server (necessarily) involved. This makes it very hard to administrate the game state such that it is consistent, and additionally there is no working business model for such a scheme.

4.1 Core selection algorithms

It is desirable to determine if the server currently hosting a region is the optimal choice. As such, we wish to make sure whether there is a server in the system that would be able to provide these players with better overall performance in terms of network delay. To accomplish this, we can use a core selection algorithm, which determines which server provides the optimal placement for that region and its players.

The core selection algorithms are devised from a graph theory perspective. Upon core selection, the core nodes may be used to administrate players that join and leave groups. Such groups can be defined and updated dynamically in an MMOG, for example, based on some area-of-interest management (like existing in the same virtual region). Typically, the core node of a group is contacted for each membership change, such that it always has the latest view. When a limited set of nodes handles the membership management, it simplifies membership updates; applications with highly dynamic groups, such as MMOGs, require fast and simple group management.

Core-based protocols work on the assumption that one or more core nodes are selected as group management and forwarding nodes. Therefore, the cores need to be selected using a core selection algorithm. Several core selection algorithms have been proposed, and a comprehensive study is given by Karaman and Hassanein [31]. An overall goal is to select cores on the basis of certain node properties, such as, bandwidth and computational power. We wish to base this decision primarily on latency. The cores that are selected depend on the group size and location, as well as the capacities in the available core nodes. In this paper, our focus is on electing a single core node for each defined group. The core may, for example, be a server or a proxy administrated by the game provider.

The core selection algorithms presented below search among a predefined set of servers and proxies to find one optimal core, which is the *graph median*. The graph median is the node for which the sum of lengths of shortest paths to all other vertices's is the smallest. The algorithms are [31]:

- *Topology Center:* Find a central entity (server) that is closest to the topological center of the *global graph*.
- *Group Center:* Find a proxy that is closest to the group center of the *group graph*.

Topology center is given as input a set of available servers that are located around the world. The algorithm searches for the server for which the sum of latencies of shortest paths to all players in its member network is the smallest, which is comparable to finding an optimal server for an instance of the game world for all connected players. The group center algorithm similarly searches among a set of available proxies located around the world. It is given a group of players as input, and based on this the algorithm selects the core proxy to be the proxy for which the sum of latencies of shortest paths to all the players in the group is smallest, i.e., locating an optimal placement for a subset of the players (interacting in a region of the game instance, for example). These simple algorithms form powerful techniques in the search for suitable hosts to migrate game state to. We have tested several algorithms [43] and present the two most prominent, in the following. Both k-Median and k-Center find optimal solutions to two different graph theoretical problems.

4.1.1 *k-median core-node selection algorithm*

The k-Median core-node selection algorithm finds k core-nodes that are the k nodes with the lowest average pair-wise distances to the nodes in the member-node set. The algorithm solves the k-minimum-pairwise problem, which when given a weighted graph $G = (V, E, c)$, and an integer $0 < k < |V|$, finds a set $D \subset V$ of size k, such that

the sum of the distances from the vertices $u \in D$ to all nodes $v \in V$ is the smallest. The k-Median algorithm has a time-complexity of $O(n^2)$ on any graph.

Algorithm 1 k-MEDIAN(G):

```
1: Input: An integer k > 0, a graph G = (V, E, c). Sets Z ⊂ V and X ⊂ V.
2: Output: A set C ∈ X of core-nodes.
3: map<id, pair-wise> mapIdPairwise
4: for each x ∈ X do
5:     T = ShortestPathTree(x, G)
6:     v = maxEccentricityNode(T, Z)
7:     pairwise = getPairwiseDistances(x, T)
8:     mapIdPairwise.insert(x, pairwise)
9: end for
10: C = kLowestPairwise(mapIdPairwise)
```

4.1.2 *k-center core-node selection algorithm*

The k-Center core-node selection algorithm finds k core-nodes that are the k nodes with the lowest maximum distance (eccentricity) to a node in the member-node set Z. It solves the k-minimum-eccentricity problem, which when given a weighted graph $G = (V, E, c)$, and an integer $0 < k < |V|$. Find a set $D \subset V$ of size k, such that the sum of the eccentricities yielded by the vertices $v \in D$ is the smallest. The k-Center algorithm has a time-complexity of $O(n^2)$, when run on a complete graph.

Algorithm 2 k-CENTER(G):

```
1: Input: An integer k > 0, a graph G = (V, E, c). Sets Z ⊂ V and X ⊂ V .
2: Output: A set C ∈ X of core-nodes.
3: map<id, eccentricity> mapIdEcc
4: for each x ∈ X do
5:     T = ShortestPathTree(x, G)
6:     v = maxEccentricityNode(T, Z)
7:     ecc = getEccentricity(v, T, Z)
8:     mapIdEcc.insert(x, ecc)
9: end for
10: C = kLowestEccentricities(mapIdEcc)
```

5 Latency estimation

One important issue related to centralized core-node selection algorithms is that they need all the required network information to be available at the executing node. As monitoring the entire network is too expensive and does not scale, latency estimation techniques are important to enable centralized core-node selection algorithms in large-scale applications, such as MMOGs. When latency estimates are available, the issue then becomes how these latency estimates affect the performance of the core-node selection algorithms, which we evaluate in Section 6.3.2. We have discussed different classes of latency estimation techniques (in Section 2.4), where we

Table 1 Properties of the latency estimation techniques

Technique	Measurement overhead	Requires	Churn recovery	Infrastructure dependability
Vivaldi	–	Inter-nodes traffic	Yes	No
Netvigator	$O(L * N)$	Traceroute	No	Yes

concluded that landmarks- and multidimensional-scaling based latency estimation techniques are the most appropriate for the problem domain. Netvigator [41] and Vivaldi [18] are two highly valued techniques in their respective latency estimation technique classes [23]. Table 1 provides a small comparison.

Vivaldi is a multidimensional scaling technique and is based on spring embedding, which models network nodes as masses connected by springs (links) and then relaxes the spring length (energy) in an iterative manner to reach the minimum energy state for the system. All nodes joining the system are placed at the origin, and start sharing Vivaldi information with selected nodes piggybacked on application level data. The Vivaldi information includes its coordinates, confidence estimations and the measured latency. If a global graph is desired, each node can report its Vivaldi information to a repository that does some calculations and inserts the node in a two-dimensional plane where the Euclidian distance equals the estimated latencies. Vivaldi has the advantage that it recovers from node churn (nodes joining and leaving), and does not depend on any infrastructure.

Netvigator, often considered the most accurate [41], on the other hand, needs landmark nodes and does not (easily) recover from churn. With Netvigator, a set of landmark nodes L are probed asynchronously by N nodes using Traceroute ($L*N$ probes). Each node reports its measurements to a repository (typically a server node), which estimates a global graph of latencies. Netvigator was originally designed for proximity estimation, that is, to rank nodes according to proximity to any given node.

6 Evaluation

In order to evaluate our system, we have performed several experiments, using simulations and live tests on PlanetLab. We first evaluate the costs of using our migration prototype. Then, we look at core selection before analyzing the network estimation accuracy and its influence on the core selection.

6.1 Migration

The migration and name service are implemented as a proof-of-concept prototype. We have tested this prototype by implementing a basic protocol for group communication, with the intention of imitating the interaction (and distribution) patterns that we would expect to see in a virtual region. The test consisted of two servers (for pre- and post-migration) and several clients. All the clients joined the same communication group at the same initial server, generating random messages at irregular intervals. After a period of time, the communication group was migrated to the post-migration server. After migration, the clients seamlessly continued their

interaction. The described scenario was run in excess of 100 times. Each time the clients were able to continue their interaction unhindered. As such, the system has shown that it is capable of migrating objects, maintain references to these objects (through the name service), and have the clients reconnect to their new location.

An important aspect of such a system is whether the overhead of migration is too large with respect to low latency communication. Therefore, to measure overhead, we have timed a remote method invocation (which includes a look up in the name service, serialization and deserialization, and a few other operations) and compared it to a regular method invocation. We ran the tests on an Intel Core 2 Duo, using only one core, which was clocked at 800MHz. The machine was running the operating system Linux (kernel version 2.6.22-14), and both the *sender* and *receiver* were both running on the same machine. The result is summarized in Table 2, where we can see the (expected) overhead of the migration middleware itself. This overhead is considerable, but negligible when compared to the overhead added by the network. Thus, with respect to the latency gain, this is totally dependent on the core node that is found.

We also need to consider the overhead of migrating a virtual region, though this will greatly depend on the number of objects being migrated, and the size of the objects in question. With our middleware, we migrate data only, as the code is shared. We give application developers the possibility of defining the parts of an object that they wish to migrate, as detailed in Section 3.3. The serialization mechanisms and more are described in further detail in [5, 6].

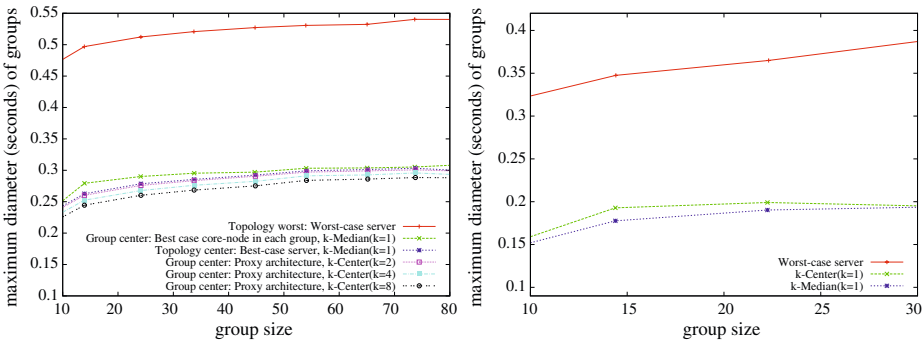
6.2 Core selection

Now that we are able to transparently migrate the game state to another node, we need to find the most appropriate node to migrate to. To test our proxy (core) selection algorithms, we have simulated several algorithms [43] mimicking group communication in a game, and we present the results from two of the most promising: *k*-median and *k*-center (Sections 4.1.1 and 4.1.2, respectively).

In our first experiment, we perform a simulation. Our network is generated using BRITE [35] with flat, undirected Waxman topologies [44] consisting of 1000 nodes. The network layout is a square world with sides equal to 200 ms. The nodes join and leave groups throughout the simulation, causing group membership to be dynamic, and group popularity is distributed according to a Zipf distribution [9]. As a metric, we use the worst-case pair-wise latency between clients (diameter) in a network (measured from the core node). Thus, the diameter should be below the latency requirements of the application (see Section 2.1), and it is desirable that the diameter is as low as possible. Figure 5a plots the average group diameter for which a single core-node selection algorithm has chosen a server-node as the root of the group tree (all communication flows via the root). We see that choosing the server to be in the topology center does significantly reduce the group diameter. It is also clear that having a limited number of proxies (other core nodes) placed around the world can

Table 2 Cost of method invocation

Invocation type	# of Invocations	Mean overhead
Remote	100,000	2.26541 ms
Normal	100,000	0.10125 μs



(a) Simulator: k number of cores. (b) PlanetLab: a single core.

Fig. 5 Diameter (seconds) of groups using core selection (**a**, **b**)

reduce the group diameter. And as expected, increasing the number of proxies does decrease the diameter further.

In our next experiment, we performed experiments on PlanetLab. Figure 5b plots the group tree diameter based on experiments done on 100 PlanetLab nodes. We observe similar results as from the simulations using the core selection algorithms to select a server, we can greatly reduce the diameter of the communication tree.

As our results show, we are able to find a suitable proxies which lowers the diameter of the distribution tree, when taking into account the available proxies and the locations of the group of players active in the give region.

6.3 Latency estimation

As mentioned above, in order to make an efficient server selection using the core selection algorithms, we need information about the network. In Section 5, we saw that full monitoring is too expensive and estimations are therefore frequently used. In this section, we therefore present results from experiments measuring the accuracy of Netvigator and Vivaldi by comparing their estimates to real all-to-all ping measurements. Then, we look at the estimates' influence on the server (core) selection algorithms. For our experiments, we again used PlanetLab using 215 nodes (the total number of nodes we were able to access). We performed latency tests over a period of 10 days.

For the Netvigator experiments, we used publicly available estimates performed on PlanetLab. Netvigator is currently a running PlanetLab service that estimates the link latencies between nearly every PlanetLab node. We used these measurements in our experiments. The Netvigator configuration is currently a black box for us.

For the Vivaldi experiments, we used a combination of the parameters in Table 3, and used *group sizes* up to 12 nodes (more neighbors makes more measurements and better estimations [18]). The RTT measurements were obtained in two different manners using *tcpinfo* or *ping* (but the results are very similar to [43] and the plots below therefore only how the tcpinfo results). The *packet rate* was varied because a higher rate follows the actual latency development more closely, while it is also consuming more bandwidth itself, at least in the active measurements. The *log times*

Table 3 Vivaldi experiment configurations

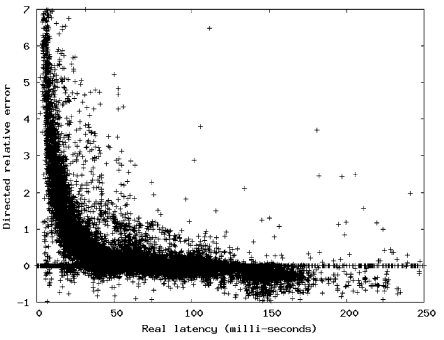
Descriptions	Configurations
Group sizes	$g = 4, 8, 12$ clients
RTT measures	*tcpinfo*, *ping*
Packet rates	High (100 packets/s), low (2 packets/s)
Log times	$t = 4, 8, 12, 16, 20$ min

(t) parameter determined for how long the Vivaldi information was collected until its estimations were used for identification decisions.

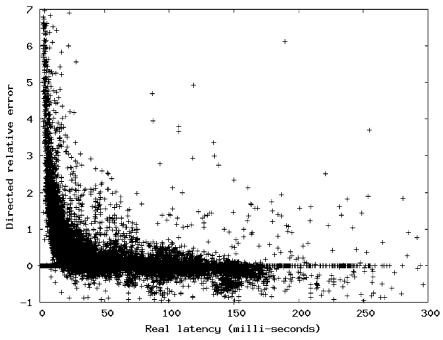
6.3.1 *Estimate accuracy*

As a measure of the estimation accuracy, we use the metric directed relative error, i.e., ping-measured all-to-all RTTs compared to the latency estimates for each pair of PlanetLab nodes (for other metrics, see [43]). A scatterplot of the results is shown in Fig. 6 plotting the directed relative deviation between the measured (real) latency versus the estimated latency, i.e., each point optimally should be on the $y = 0$ line.

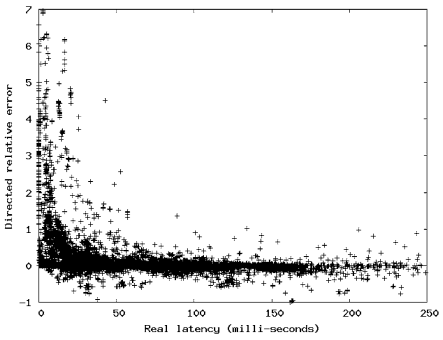
We see that Netvigator is very accurate in its estimations, closely following the ideal line whereas Vivaldi has a bit more variation. Netvigator yields 80% of the estimations within a 15% relative error of the ping measurements and is clearly best. Vivaldi estimations are best in configurations with a high packet rate, and yields 80%



(a) Vivaldi, low packet rate.



(b) Vivaldi, high packet rate.



(c) Netvigator.

Fig. 6 Directed relative error of latency (**a**–**c**)

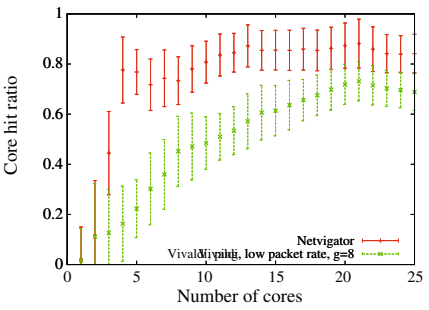
of the estimations within a 50% relative error for a 4-min log time. This is because more probes generates more statistics for Vivaldi to measure from in a shorter time-span. Lower packet rates require a considerably higher log time to approach the relative error satisfyingly. Furthermore, both Vivaldi, and to a lesser extent, Netvigator, overestimate RTTs for the smaller actual RTTs, while underestimating for longer distances. When the actual RTTs are very small, the overestimations are relatively high, but the absolute deviation may still be acceptable for many applications.

6.3.2 *Latency estimates applied to core-node selection*

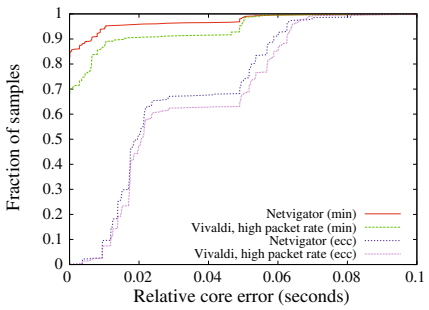
The question now is how the estimation (in)accuracies influence the core selection algorithms. The following results are gathered by applying the Netvigator and Vivaldi latency estimates to the core-node selection algorithm k-Median, $1 < k \leq 25$. For the experiment, we used 100 nodes from PlanetLab. The core nodes were found using the network information obtained using both the estimation techniques and all-to-all measurements. For the Vivaldi estimates, we allowed a period of 4 minutes to let the node coordinates stabilize. The tests were run each day for a 10 day period.

Figure 7a plots the core-node selection hit ratio, which measures the ratio of "hits" for each time the k-Median finds the same core-node using estimated latencies and real all-to-all ping measurements. It is clear that Netvigator yields better estimates for use in core-node search, and stabilizes around 80% core-node selection hit ratio quickly. The hit-ratio is much lower using Vivaldi estimates (using a high packet rate gives slightly better results), especially when the number of core-nodes is less than 10.

Figure 7b plots the CDF of the minimum (min) core-node error in terms of latency between the core-nodes found using latency estimates and the (optimal) core-nodes found using the real all-to-all ping measurements. If the core-node error is zero, the same core-node(s) is found using estimates and real measurements. As expected, k-Median is most accurate when it uses Netvigator, with 95 % of the core-nodes within 10 ms of an optimal core-node. The Vivaldi estimates makes k-Median return more inaccurate results, with 85 % of the core-nodes within 10 ms of an optimal

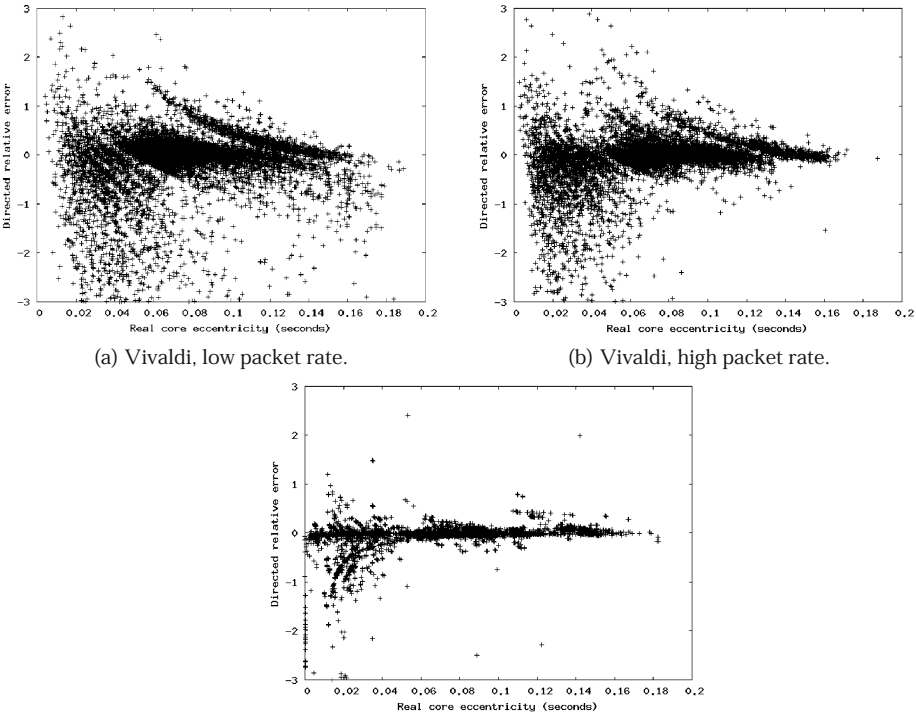


(a) Ratio of optimal core selection hits when k-Median is applied to Vivaldi and Netvigator estimates.



(b) CDFs of relative core error. Minimum (min) latency and eccentricity (ecc) for cores from estimates to real cores.

Fig. 7 Netvigator and Vivaldi performance in core selection (**a**, **b**)



(a) Vivaldi, low packet rate.

(b) Vivaldi, high packet rate.

(c) Netvigator.

Fig. 8 Directed relative error of the reported eccentricity to the real eccentricity from cores to group members (**a**–**c**)

core-node. Moreover, Fig. 7b also plots the maximum latency (ecc) between the core-nodes found using latency estimates and the optimal core-nodes. Finally, the discrepancy between the reported eccentricities and the real eccentricities are plotted in Fig. 8 in terms of the directed relative error. Netvigator is clearly the best and has a performance very close to the real eccentricity.

In summary, it is clear that Netvigator's more accurate estimates enable the core-node selection algorithm to return the better core-nodes. Nevertheless, Vivaldi's estimates still enable the core selection to find nodes within 20 ms of the optimal core-nodes. Thus, both estimation techniques could be used. An important note, however, is that the performance of Vivaldi depends on the packet rate, and in the game scenario, the packet rate is closer to the tested low rate. Piggybacking Vivaldi data in the data packets might therefore not be sufficient and additional probing packets might be required.

7 Discussion

Following the discussion throughout this paper, a few things become apparent. In order to perform migration we need a way of maintaining references to objects. As such, any object is identified through the name service by querying its

home-address, an identifier which is provided by the name service at object creation. The indirection through the name service is necessary for accessing objects at remote nodes transparently. Core selection is dependent on full knowledge of the network, but obtaining latency measurements through active probing is not a scalable solution. Latency estimation techniques provide an alternative.

7.1 Implications of migration

Due to the distributed name service resolving references to remote objects can become a time consuming task. An object that is frequently migrated will create trails of object references at the nodes it visits. To reach the object, one could be required to unravel a number of these references before being able to access the object, though this will depend on where in the chain of references the object request propagates from. One possible solution to solving this problem is to use leases, as originally described by Gray et al. [28]. They describe a lease as a promise from the server that it will push updates to the client for a specified amount of time. A more flexible approach is described by Duvvuri et al. [21] in form of adaptive leases, which make it possible for the server to adapt leases depending on different criteria. Thus, the node that has migrated the object can request a lease on an object, and have the server send it updates about the object reference for the duration of that lease.

Another concern is related to the side-effects of migrating game-state during execution of the simulation. If migration is not performed transparently, we might adversely affect the interacting players. In [36], Nelson et al. demonstrate how an active application is moved from one virtual machine to another, with minimal perceived impact to the user's interaction with the application.

We also need to consider how to clean-up objects that are no longer in use. With a distributed system, where objects are frequently moved around, the garbage collection process becomes more complicated then otherwise. One solution is to use reference counting, when the number of references to an object reaches zero, we can remove the object. As such, each object holding a reference to another object must send a message when it is no longer interested in holding a reference.

7.2 Partial failures

Partial failures occur when a node in a distributed system becomes unavailable, effectively rendering the objects managed by it inaccessible. The current architecture does not accommodate for partial failures, but there are ways to minimize the repercussions of these incidents. One possibility has its roots in peer to peer based file systems, where copies of an object will be distributed to several nodes in the system. PAST [40] and OceanStore [27] have, for example, implemented such systems with success. PAST copies objects to random nodes, in an attempt to distribute the objects evenly. OceanStore uses a more deterministic approach, and places the objects close to nodes which access them. Lookup of objects in the system can then be implemented in a fashion similar to that of Chord [42] or Tapestry [46]. These implementations are based on the principle of incrementally forwarding messages from point to point, until they reach their destination. Each node in the system keeps a small routing map, which is used to determine which nodes to forward the message to. A problem with this type of look up is that the response time might be too high

for interactive applications. A centralized approach would avoid the lookup time problem, but would itself become a point of failure, and would also have to handle all the network messages, which could potentially congest the server.

7.3 Latency estimation techniques

We have evaluated Netvigator and Vivaldi as popular representatives for the latency estimation techniques. It is clear that Netvigator yields better estimations, but it is also more difficult to set up as it depends on landmark nodes. In the scenario of MMOGs, when the game provider has dedicated servers, this is not an issue, the benefit of the increased accuracy outweighs the initial setup cost. However, while Vivaldi does perform worse, it has the advantage of easy deployment and the ability to recover from churn. The Netvigator configuration is a blackbox, as PlanetLab has not disclosed this information. For Vivaldi, however, the best configuration was a group size of 8 (and above) and high packet rates. A low packet rate reduced the estimation accuracy, and required 8 min to stabilize, in contrast to 4 min at high packet rates. This might be important to keep in mind, as Vivaldi piggybacks probing information, and many games have a low packet rate [38], i.e., additional probing packets might be needed using Vivaldi. For both Netvigator and Vivaldi, k-Median is able to find close-to-optimal core-nodes, and it is clear that Netvigator estimates enable the core-node selection algorithm to return the more optimal core-nodes. Nevertheless, Vivaldi's estimates still enable the core selection to find nodes within some tens of milliseconds distance of the optimal core-nodes (with the server densities that we tested). Thus, both estimation techniques could be used.

7.4 Activation policies

Core selection and migration require resources in the form of processing power and network messages when activated. As such, they should be triggered methodically and on-demand, with policies governing their activation. The combined process consists of two stages, where the first stage evaluates if activating a core selection is potentially beneficial. The activation criteria we have identified so far are: churn (in the form of players joining/leaving), time of day, player arrival rate, history based (this being a preemptive approach, where known situations, such as battles, trigger the activation) and server-load. If the first stage is completed successfully, and a new core is selected, the second stage needs to determine if performing a migration is beneficial. Metrics of consideration include: threshold of aggregate latency, number of players, packet loss, and server load. For both stages the metrics are not necessarily mutually exclusive, and we do not consider these lists to be exhaustive.

8 Conclusion

It has been shown that there is a strong correlation between latency and the playability of an online game [16], with the perceived game play deteriorating considerably as the latency increases.

An important factor for world-spanning games, such as MMOGs, lies in the diversity of its user base. There will, at any point in time, be a number of players connected

from different physical locations. It has, however, been shown that distinct groupings will appear, and change with the time-of-day [24, 29]. There have been made few efforts into determining how to best support the dynamic player masses in virtual worlds hosting thousands of concurrently interacting players, when geographically distributed servers are available.

In this paper, we have presented a viable solution with geographically distributed servers. We have shown how core selection can be used to find an optimal node in the system for placing a virtual region, and correspondingly the players interacting in that region. We have looked at how latency estimation techniques can be utilized to gather information about the network in a scalable manner. Once an optimal node has been located we can migrate the game state to that node, maintaining references to the migrated state through the use of our distributed name service. By performing this migration, the overall latency of that region can be lowered decreasing the average network latency.

Thus far, we have implemented the migration and name service as a proof-of-concept, and run some basic tests to determine its usability. Furthermore, we have run simulations on core selection and determined its applicability in the scenarios we have described. Additionally, we have run extensive tests that show how latency estimation, using for example Vivaldi or Netvigator, provides a scalable approach to gathering information about the network. Now, it remains to integrate this functionality with an application and run large-scale tests.

References

1. Armitage G (2008) Client-side adaptive search optimisation for online game server discovery. Lect Notes Comput Sci 4982:494
2. Armitage G (2008) Optimising online fps game server discovery through clustering servers by origin autonomous system. In: International workshop on network and operating system support for digital audio and video (NOSSDAV)
3. Barak A, Guday S, Wheeler RG (1993) The MOSIX distributed operating system: load balancing for UNIX. Springer, New York
4. Beigbeder T, Coughlan R, Lusher C, Plunkett J, Agu E, Claypool M (2004) The effects of loss and latency on user performance in unreal tournament 2003. In: The proceedings of NetGames'04, Portland, pp 144–151
5. Beskow P (2007) Migration of objects in a middleware for distributed real-time interatctive applications. Master's thesis, Department of Informatics, University of Oslo, Norway
6. Beskow P, Halvorsen P, Griwodz C (2007) Latency reduction in massively multi-player online games by partial migration of game state. In: Second international conference on internet technologies and applications, Wrexham, pp 153–163
7. Beskow P, Vik K-H, Griwodz C, Halvorsen P (2008) Latency reduction by dynamic core selection and partial migration of game state. In: Proceedings of NetGames'08, Worcester
8. Blair G, Coulson G, Robin P, Papathomas M (1998) An architecture for next generation middleware. In: Proceedings of the IFIP international conference on distributed systems platforms and open distributed processing. Springer, Berlin Heidelberg New York, pp 191–206
9. Brookes B (1968) The derivation and application of the Bradford-Zipf distribution. J Doc 24(4):247–265
10. Brun J, Safaei F, Boustead P (2006) Server topology considerations in online games. In: NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on network and system support for games. ACM, New York, p 26

11. Chambers C, Feng W, Feng W, Saha D (2003) A geographic redirection service for on-line games. In: Proceedings of the eleventh ACM international conference on Multimedia, Berkeley, pp 227–230
12. Chambers C, Feng W, Sahu S, Saha D (2005) Measurement-based characterization of a collection of on-line games. In: The proceedings of the 5th ACM SIGCOMM workshop on internet measurement, Berkeley, pp 1–14
13. Clark C, Fraser K, Hand S, Hansen JG, Jul E, Limpach C, Pratt I, Warfield A (2005) Live migration of virtual machines. In: NSDI'05: proceedings of the 2nd conference on symposium on networked systems design & implementation. USENIX Association, Berkeley, pp 273–286
14. Claypool M (2005) The effect of latency on user performance in real-time strategy games. Elsevier Comput Netw 49(1):52–70
15. Claypool M (2008) Network characteristics for server selection in online games. In: Proceedings of the fifteenth annual multimedia computing and networking (MMCN'08), vol 6818, San Jose, p 681808
16. Claypool M, Claypool K (2005) Latency and player actions in online games. Commun ACM 49(11):40–45
17. Costa M, Castro M, Rowstron A, Key P (2004) Pic: practical internet coordinates for distance estimation. In: ICDCS '04: proceedings of the 24th international conference on distributed computing systems (ICDCS'04). IEEE Computer Society, Washington, DC, pp 178–187
18. Dabek F, Cox R, Kaashoek F, Morris R (2004) Vivaldi: a decentralized network coordinate system. In: ACM international conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM), pp 15–26
19. Dick M, Wellnitz O, Wolf L (2005) Analysis of factors affecting players' performance and perception in multiplayer games. In: The proceedings of NetGames'05, Hawthorne, pp 1–7
20. Dovrolis C, Ramanathan P, Moore D (2001) What do packet dispersion techniques measure? In: INFOCOM 2001. Twentieth annual joint conference of the IEEE computer and communications societies, Proceedings, vol 2. IEEE, Piscataway
21. Duvvuri V, Shenoy P, Tewari R (2003) Adaptive leases: a strong consistency mechanism for the world wide web. IEEE Trans Knowl Data Eng 15(5):1266–1276
22. Egenhofer M, Spatial S (1994) A query and presentation language. IEEE Trans Knowl Data Eng 6(1):86–95
23. Elmokashfi A, Kleis M, Popescu A (2007) Netforecast: a delay prediction scheme for provider controlled networks. In: IEEE Globecom
24. Feng W, Feng W (2003) On the geographic distribution of on-line game servers and players. In: The proceedings of NetGames'03, Redwood City, pp 173–179
25. Feng W, Chang F, Feng W, Walpole J (2002) Provisioning on-line games: a traffic analysis of a busy Counter-strike server. In: The proceedings of the 2nd ACM SIGCOMM workshop on internet measurement, Marseille, pp 151–156
26. Funcom (2008) Anarchy online. http://www.anarchy-online.com/
27. Geels D (2002) Data Replication in OceanStore. Tech. Rep. UCB//CSD-02-1217, Computer Science Division, U. C. Berkeley
28. Gray C, Cheriton D (1989) Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. SIGOPS Oper Syst Rev 23(5):202–210
29. Griwodz C, Halvorsen P (2006) The fun of using TCP for an MMORPG. In: International workshop on network and operating system support for digital audio and video (NOSSDAV). ACM, New York, pp 1–7
30. Jul E, Levy H, Hutchinson N, Black A (1988) Fine-grained mobility in the Emerald system. ACM Trans Comput Syst (TOCS) 6(1):109–133
31. Karaman A, Hassanein HS (2006) Core-selection algorithms in multicast routing—comparative and complexity analysis. Comput Commun 29(8):998–1014
32. King B (2007) GCC-XML the xml output extension to gcc. Undated. http://www.gccxml.org/HTML/Index.html
33. Kupfer MD (1993) Sprite on mach. In: MSYM'93: proceedings of the 3rd conference on USENIX MACH III Symposium. USENIX Association, Berkeley, pp 7–7
34. Lee K, Ko B, Calo S (2005) Adaptive server selection for large scale interactive online games. Comput Netw 49(1):84–102
35. Medina A, Lakhina A, Matta I, Byers J (2001) BRITE: Universal topology generation from a user's perspective. Tech. rep. BUCS-TR-2001-003, Computer Science Department, Boston University

36. Nelson M, Lim B, Hutchins G (2005) Fast transparent migration for virtual machines. In: Proceedings of the USENIX annual technical conference 2005 on USENIX annual technical conference table of contents, pp 25–25
37. Ng T, Zhang H (2001) Towards global network positioning. In: Proceedings of the 1st ACM SIGCOMM workshop on internet measurement. ACM, New York, pp 25–29
38. Petlund A, Evensen K, Griwodz C, Halvorsen P (2008) Improving application layer latency for reliable thin-stream game traffic. In: Workshop on network and system support for games (NETGAMES), pp 91–98
39. Powell ML, Miller BP (1983) Process migration in demos/mp. Tech. rep., Berkeley
40. Rowstron A, Druschel P (2001) Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In: Proceedings of SOSP'01, Lake Louise, pp 188–201
41. Sharma P, Xu Z, Banerjee S, Lee S-J (2006) Estimating network proximity and latency. SIGCOMM Comput Commun Rev 36(3):39–50
42. Stoica I, Morris R, Karger D, Kaashoek M, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In: Proceedings of SIGCOMM' 01, San Diego, pp 149–160
43. Vik K-H (2008) Group communication techniques in overlay networks. Ph.D. thesis, Department of Informatics, University of Oslo, Norway
44. Waxman BM (1991) Dynamic Steiner tree problem. SIAM J Discrete Math 4:364–384
45. Wong B, Slivkins A, Sirer E (2005) Meridian: a lightweight network location service without virtual coordinates. Computing and Information Science Technical Report TR2005-1982, Cornell University
46. Zhao B, Kubiatowicz J, Joseph A (2001) Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, Computer Science Division, U. C., Berkeley
47. Znati TB, Molka J (1992) A simulation based analysis of naming schemes for distributed systems. In: Proceedings of the 25th annual simulation symposium, Los Alamitos, pp 42–53

Paul B. Beskow received his B.Sc. in Informatics in 2003, and his M.Sc. in 2007, both at the University of Oslo (UiO) and is currently working on his Ph.D. at UiO and part-time at Simula Research Laboratory. His main research interests include system support for low-latency communication including object migration, network protocol optimizations, as well as various TCP protocol optimization issues.

Knut-Helge Vik is a Ph.D. student at the Simula Research Laboratory. His recent research activities address system demands for distributed interactive applications. The research include graph theoretical problems for overlay network design, practical simulations and experiments of overlay network construction algorithms, group management requirements related to latency and consistency, etc. He received a Master of Science in Computer Science from Washington State University in 2004, and has been a Ph.D. student at the Department of Informatics, University of Oslo and Simula Research Laboratory since October 2004. More information and the publication list can be found at http://home.ifi.uio.no/knuthelv.

Pål Halvorsen is an associate professor at Simula Research Laboratory and at the Department of Informatics, University of Oslo, Norway. He is member of the Center for Research-based Innovation "Information Access Disruptions". He received his master and doctoral degree in computer science from the University of Oslo in 1997 and 2001, respectively. His research activities focus mostly on resource utilization and system support for distributed multimedia systems, and in particular, in the area of on-demand streaming applications and multiplayer games.

Carsten Griwodz is a professor at Simula Research Laboratory, Norway and at the Department of Informatics at the University of Oslo. He is member of the Center for Research-based Innovation "Information Access Disruptions". He received his Diploma in Computer Science from the University of Paderborn, Germany, in 1993. From 1993 to 1997, he worked at the IBM European Networking Center in Heidelberg, Germany. In 1997 he joined the multimedia communications lab at Darmstadt University of Technology, Germany, where he obtained his doctoral degree in 2000. His interests lie in the improvement of system support for interactive distributed multimedia, with operating systems and protocol support for on-demand streaming applications and multiplayer games in particular.