Latency Reduction by Dynamic Core Selection and Partial Migration of Game State

Paul B. Beskow, Knut-Helge Vik, Pål Halvorsen, Carsten Griwodz
Simula Research Laboratory, Norway IFI, University of Oslo, Norway
{paulbb, knuthelv, paalh, griff}@ifi.uio.no

ABSTRACT

Massively multi-player online games (MMOGs) require low latency while supporting a large number of concurrent players, often sharing one worldwide instance of the game. As these are conflicting requirements, a common way of distributing load is by dividing the virtual environment into virtual regions. As MMOGs are world-spanning games, it is plausible to disperse these regions on geographically distributed servers. As such, we propose the use of core selection for finding an optimal server for placing a region, and support for migrating the game state to that server. The first goal relies on a set of servers and measurement of the interacting players latencies. In locating an optimal server, we anticipate a decrease in the overall latency for the majority of players. This reduction occurs by migrating the region to a server closer in proximity to the majority of players in that virtual region, thereby lowering the response time of any interaction.

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*. It should be understood that this term holds no positive connotation, as a player that has high latency will inadvertently have a negative effect on the perceived quality of the game play [3, 10]. This occurs, as most online games are based on a client-server model, where events are collected at the server, and accordingly 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, but will propagate to other interacting parties, potentially resulting in inconsistencies. While having minor effects on the outcome of the game, it results in a perceived deterioration to the quality of interaction [14]. As such, *low latency* for *all players* is a prevalent goal.

As latency is affected by physical distance between the player and server, reducing the latency to all players is an arduous task. Thus, providing the required level of interactivity, within the latency requirements of the game (ranging from 100 to 1000 ms [12]), may demand that the player is in *close proximity to the server*.

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, this as the regions are logically decomposed, i.e., each responsible for handling some regions and the players interacting in each region. As the geographical dispersion of players in MMOGs depends heavily on the time of day [9], it is possible to find players clustered by their virtual (belonging to the same region) as well as physical location. Since a player's proximity to the server impacts the latency, and in turn, since 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 and its interacting players given a set of globally distributed servers?*.

In the remainder of this paper, we will describe our use of core selection for finding the most appropriate server (or proxy) for placing a virtual region, when given a player base and a set of available servers. 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 with the combination of *core selection, a distributed name server and migration* we have a viable solution for creating a globally distributed game, which is capable of *lowering the overall latency of the interacting players.*

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 [7, 8], 2) some latency is tolerable [10], as long as it does not exceed the threshold for playability, i.e., ranging from 100ms to 1000ms depending on the type of game [12] and 3) geographical dispersion of players in an online game depends heavily on the time of day [9]. In addition to these works, we have analyzed packet traces (see [17]) from Funcom's popular role-playing MMOG *Anarchy Online* (AO) [16]. Statistics from the traces reveal that there is a potential gain with respect to latency by using our migration middleware. For example, in one of the game regions within a timespan of about one hour, we found approximately 175 distinct connections. These are sorted according to their measured RTT in figure 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.

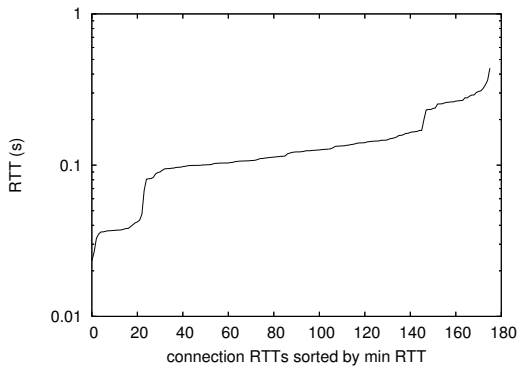


Figure 1: AO: connection RTTs sorted by min RTT

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, i.e., as the majority of users are in the second range in figure 1(20-140), the average latency could be reduced by moving the game region to a server in Europe.

2.2 Server selection

Game server selection is an important facet for the playability of a game. 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. To this respect, Chambers et al [6] have looked at how server selection can be optimized for a single client, when given a set of available servers. Following this scenario and figure 2, where the circles denote acceptable boundaries for playability, we see that the servers in *Chicago* and *New York* will be weighted equally for *Jane*, while only *New York* is acceptable for *Anette*. In a further study, Claypool [11] 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. Given this scenario and figure 2, the guided selection process would weight *Chicago* higher than *New York* for *George*, *Jane* and *Tom*, while *New York* is weighted the highest for *Jane* and *Anette*. A third study, which is related to these topics, looks at how the search itself can be performed in an optimal way [1]. The two former papers both have in common that they consider server selection from the perspective of the player(s), and additionally assume a certain availability of servers. As we can see from figure 2, it is common for geographically coupled users 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. Thus, the differences in geographical locality become more apparent. When we consider the time-of-day characteristics of geographical dispersion, it would be beneficial to examine if these techniques can be applied by a server, to improve the playability for a group of players, e.g., the players in the same virtual region.

2.3 Virtual regions

As MMOGs are expected to handle thousands of concurrently interacting players it is common practice to use a static, region based partitioning scheme. The virtual environment is thus divided into smaller, more manageable parts, where each virtual region is hosted

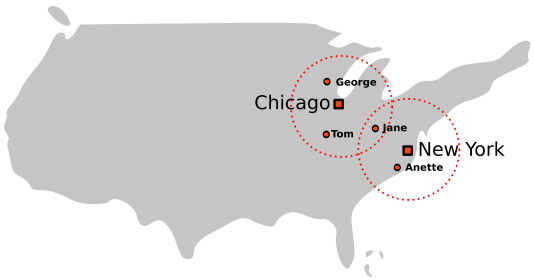


Figure 2: Clustering in FPS style games.

on a single server in the cluster. A widely accepted problem with the static partitioning scheme is that it does not take into account the dynamic nature of MMOGs. Even if the static partitioning is based on population density trends, and arranged to accommodate this, it is still susceptible to imbalances due to unforeseen events. Thus, a lot of research focuses on how to improve the flexibility of these partitioning schemes, and consequently, algorithms for efficiently distributing entities and regions. This research, however, does not address how the locality of the users, in relation to a server, will effect latency. For example, Turck et al [13] have investigated the effects of dividing a game world into dynamic micro-cells. A study with a similar background is performed by Duong et al [15]. Such micro-cells can be reassigned to servers in a cluster if the load on the server they are currently residing on becomes too large. Different load-balancing algorithms were applied, none of which factored in locality of users, and the number of micro-cells supported per server was varied. The test was performed on a centralized cluster. A similar approach is deployed by IBM with the Matrix [2] middleware. Here, the focus is on the player, where consistency updates are limited to an *area of interest*. It is based on the observation that MMOGs are nearly decomposable systems, and as such, it is usually sufficient to update players only with those events that occur within their zone of visibility. Matrix makes use of region based partitioning as an underlying foundation, but this is for the purpose of easily distributing the virtual world across multiple servers. Consistent for all these approaches is that a dynamic solution outperforms a static one.

2.4 Summary

In summary, the work on static and dynamic partitioning consider server load in a centralized cluster and not latency due to the geographical location of users, this despite the fact that there is visible evidence of such a trend. Most of the research tries to optimize the partitioning of the virtual environment into regions which can dynamically accommodate hot-spots. Furthermore, with respect to server selection, previous work look at how to best select the best game instance, for example, with lowest latency. However, in these tests, little or no efforts have been made to investigate the effects of a decentralized distributed system middleware, which would allow for regions of the game to be migrated based on the physical locality of the users, in addition to their virtual locality. Our analysis of the game traffic shows that there are players connected from all around the world (see figure 1), but the amount of players from a part of the world will depend on the time of day. Thus, as there is a shift in the location of the majority of players, another 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 through core selection, which helps us determine which node to migrate the players to.

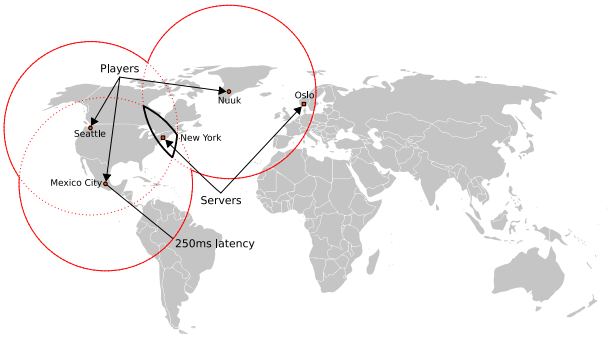


Figure 3: Latency as physical distance

3. CORE SELECTION

An efficient means to reducing latency can be accomplished by migrating game state to an appropriate server. This appropriate server may be a server close to the center of a given group of players. As an example, consider an MMOG with geographically distributed servers (see figure 3). 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. Given a region, with a number of interacting players, it is probable that a majority of these players are in reasonable proximity of each other (as seen from a geographical perspective at a given time). Furthermore, looking at figure 3, we can see how a latency requirement can be translated into a measurement of physical distance. In this example, we have two server nodes, one located in *New York* and one in *Oslo*. In addition, we have three players that are connected from *Mexico City*, *Seattle* and *Nuuk*. In case of a 250 ms delay requirement from the server (e.g, according to the 500 ms pairwise latency requirement in RTS [10]), the bordered intersection indicates an optimal area for the server to be located, and given the set of available servers, the natural choice would be to select the server in *New York* as the core server node.

Accordingly, it is desirable to determine if the server currently hosting a region is the optimal choice. In this context, optimal will be determined by the overall latency experienced by the players currently interacting there. As such, we wish to ascertain 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. Looking at figure 3, with server nodes in *Oslo* and *New York*, we can naturally assume that the core selection process would have selected *New York* as the optimal location.

To accomplish this, we can use core selection techniques, which in this scenario requires complete information about the available server nodes, and players interacting in the region. Based on this information, the heuristic core selection method will determine which server provides the optimal placement for that region and its players.

3.1 Core selection heuristics

The core search is conducted by core selection heuristics that 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.

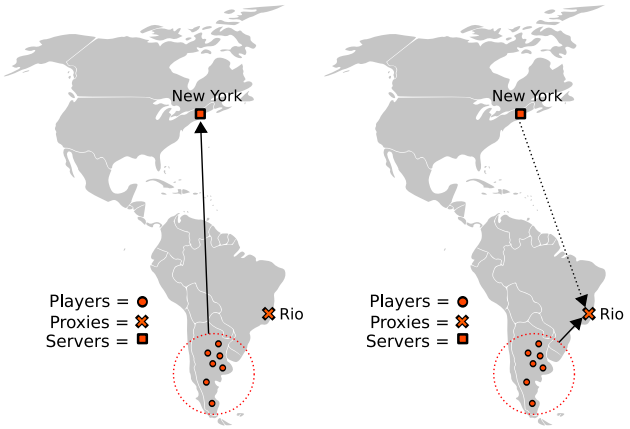
Core-based protocols work on the assumption that one or more core nodes are selected as group management and forwarding nodes. Therefore, the cores need to be selected using some core selection heuristic. Several core selection heuristics have been proposed, and a comprehensive study is given by Karaman and Hassanein [18]. 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.

3.2 Core selection in a proxy architecture

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 heavily depends on the physical distance from each individual player to the server. In figure 4(a), we illustrate an example where a centralized server stores the game state and 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, of which some are expected to be physically closer to the players than a central server. In figure 4(b), 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. Where using a pure peer-to-peer architecture implies distributing the game state among the players, using no central server. 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.



(a) Centralized architecture (b) Proxy architecture

Figure 4: Architecture

The core selection heuristics presented here 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 heuristics are [18]:

- *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*.

The topology center heuristic is given as input a set of available servers that are located around the world. The heuristic 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, i.e, for all connected players. The group center heuristic similarly searches among a set of available proxies located around the world. It is given as input a group of players, and based on this the group center heuristic 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 heuristics form powerful techniques in the search for suitable hosts to migrate game state to.

4. MOVING WORLDS WITH MIGRATION

The core selection process is responsible for determining 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, the MMOG will move 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 will be 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 latency is accomplished by decreasing the response time of, for instance, remote method invocations (RMI). This because we move objects closer to the majority of the players.

4.1 A name service

The name service maintains references to the objects in the distributed system, this goal can be accomplished in several ways. Znati and Molka [20] analyzed three approaches to implementing a name service; in the form of centralized, hybrid and distributed versions. Prior to contacting the target object itself, the centralized version contacts a name service, located at a well-known server, to obtain the objects location in the network. As such, the centralized naming scheme adds an extra level of indirection to the name resolution process. The hybrid approach is based on the design principle of keeping names together with the objects they are bound to on the local level, but resorts to multicasting when resolving names at a regional level. The distributed paradigm removes this level of indirection by placing the name of the object with the object itself. Once an object resolution has been performed, the object is accessed directly at the server managing the object. The results showed that the centralized model could achieve acceptable performance only as long as the ratio of remote to local requests was kept reasonable. The performance of the hybrid model highly depended on the efficiency of the cache design. With all other network conditions set equal they found that, relative to the response times of the centralized simulation, the response time of the distributed simulation were smaller. We relate these three models to MMOGs with geographically distributed servers by coupling them with the following characteristics:

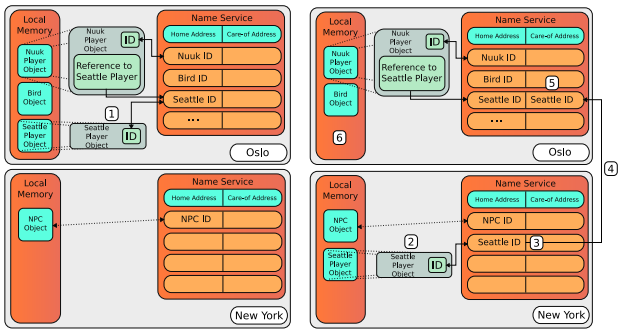
1. There are thousands of concurrently interacting players.
2. The virtual environment is divided into virtual regions.
3. Players are dispersed physically as well as virtually.
4. The physical player distribution depends on time of day.
5. The servers in the system are geographically distributed.
6.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

Given these characteristics a distributed name service best suits our needs, primarily because efficiency is more important then consistency in this scenario. It is more efficient because there is no overhead in binding an object with the name service, as this occurs locally. Given that the servers in the system are geographically distributed this becomes essential. 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 will also be efficient, as we can directly query the node our name service has registered as the current maintainer. Though it is worth nothing that this access time will depend on how many times an object has been migrated (from 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, references and migration see [4]. In the following section, we will solidify our understanding of the described mechanisms by looking at an example of them in use.

4.2 In action

Consider a system consisting of two servers, as seen in figure 3. In this scenario, it is evening in Europe, and accordingly the majority of the interacting players are European. As such, all of the virtual regions are currently being hosted in *Oslo*. In addition to the European players, we have a couple of players connected from *Nuuk* and *Seattle*. Looking at figure 5(a), we see that these two players have been bound to the name service and received an identifier (the other players are not shown in this example). We can also see that the *Nuuk* player references the player in *Seattle*. Notice that this reference is not to the player object in local memory, as we might expect, but to the identifier (created earlier) in the name service. As the time passes the population on the server shifts from being predominantly European to American. As such, the core selection process is performed for each virtual region. For some of the regions, it is found that the server in *New York* is best fit to serve the currently connected players. As we can see from figure 5(a), the region with the players connected from *Nuuk* and *Seattle* is marked for migration. At this point in time, the region these two players are interacting in is migrated to the server in *New York*. We will see how this is accomplished in our middleware, when we follow the steps outlined in figure 5(a), which denotes the actions before migration, and figure 5(b), which denotes the actions taken during and after migration.

The *first* step (1), consists of serializing the player object. Serialization is the act of creating a binary representation of the object



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

Figure 5: Server configuration

that can be transmitted across the network. In this case, the player object is migrated to our server in New York. The *second* step (2) consists of recreating the object at the receiving side, which is also known as deserialization. Once the object has been recreated, it is bound with the local name service, which is the *third* step (3). At this point in time, the Oslo server knows that the player object is in New York (it sent it there after all), but will have no way of forwarding requests, as it will have no way of knowing what object in New York to forward requests to. This is because the player object has been given a new identifier in New York, and, in effect, has no knowledge of its past history from the server in Oslo. This is why we in the *fourth* step (4) need to return the identifier assigned to the player object by the server in New York. 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). The reason for this is that any invocations made to the player object will now be intercepted correctly, because the name service in Oslo now knows the identity of the object in New York.

5. EVALUATION AND DISCUSSION

We implemented our core (proxy) selection heuristics in a simulator that mimics group communication in a game using a preselected central entity to handle the membership management. The central entity is always selected using the core selection heuristic topology center. Furthermore, we generated network topologies with the BRITE Internet topology generator [19], and in our experiments, we used flat, undirected Waxman topologies [**?**]. From this network, we created a fully connected undirected mesh, representing application layer communication that is shortest-path routed. The network size was 1000 nodes, and all the nodes join and leave groups throughout the simulation, causing group membership to be dynamic. Group popularity was distributed according to a Zipf distribution [5]. The network layout is a square world with sides equal to 200 ms. In figure 6(a), we can see how one group of players (highlighted by a dashed border) will be evaluating the (encircled) proxies, by calculating the average pair-wise latency to each proxy. In figure 6(b), we see how the core selection process has determined which proxy has the lowest average pair-wise latency, and how each player now has connected to that proxy.

Our goal is to illustrate the significance of server location. In addition, we want to show that having a proxy architecture with a limited amount of proxies can dramatically reduce the overall latencies. We can express the worst-case pair-wise latency between clients in a network through the diameter (measured from, e.g, a

(a) Setup (b) Completion

Figure 6: Pass for a group in the core selection process

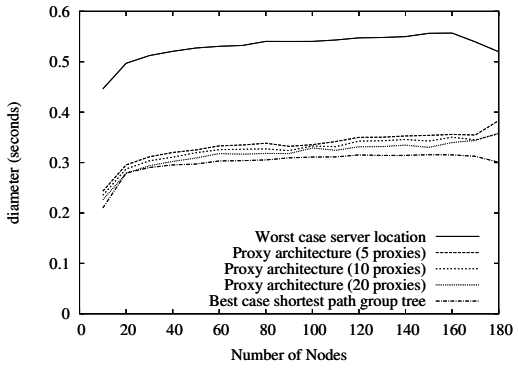


Figure 7: Diameter of group trees using a proxy architecture

core node), and it is desirable that the diameter is as low as possible. Figure 7 shows the diameter of shortest path trees for different group sizes where servers and proxies are the roots of the trees. We can see that having a limited amount of proxies placed around the world can reduce the diameter dramatically. And as expected, increasing the number of proxies will incrementally decrease the diameter. The worst-case placement of a server is not an invalid situation, because if a single server administrates the entire world the worst-case placement of a server will happen for a large number of the clients as the day passes.

As our results show, we are able to find suitable proxies which will lower 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.

The described middleware has been implemented and tested as a proof of concept. We tested the middleware by implementing a basic protocol for group communication, with the intent of imitating the interaction we would expect to see in a virtual region. Running two servers, we had several clients connect to one of the active servers, where they initiated their communication. As the clients were interacting we migrated the region to the second server, where the clients were able to continue their interaction unhindered. As such, the middleware has shown that it is capable of migrating objects, maintain references to these objects, and reconnect the interacting clients so they can continue their interaction.

To measure overhead, we have timed a remote method invocation

Invocation type	# of Invocations	Mean
Remote	100,000	2.26541 *milliseconds*
Normal	100,000	0.10125 *microseconds*

Table 1: Cost of method invocation.

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 operating system running Linux kernel version 2.6.22-14. During these tests the *sender* and *receiver* were both running on the same machine. The result is summarized in table 1. These tests represent the expected overhead added by the middleware itself for invoking methods on a migrated object, which involves name service look up, serialization, deserialization, and some additional operations. The test methods took *void* arguments and returned an *int*. The overhead of a remote invocation is considerable when compared to a regular invocation, but is negligible when compared to the overhead added by the network. With respect to the latency gain, this is totally dependent on the core node which is found. The results of these tests are reported above.

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. Though this can be accomplished transparently, without the player ever noticing. With our middleware we migrate data only, as the code is shared. We give the application developer the possibility of defining what parts of an object they wish to migrate. The serialization mechanisms and more are described in further detail in [4].

6. CONCLUSION

It has been shown that there is a strong correlation between latency and the playability of an online game [12], 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 [9, 17]. There have been made few efforts into determining how best to 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. 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, i.e., the decrease in average network latency will by far outweigh the increased overhead of remote method invocations.

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. Now, it remains to integrate this functionality with an application and run large-scale tests.

7. REFERENCES

[1] ARMITAGE, G. Optimising online fps game server discovery through clustering servers by origin autonomous system. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* (May 2008).

[2] BALAN, R. K., EBLING, M., CASTRO, P., AND MISRA, A. Matrix: Adaptive middleware for distributed multiplayer games. In *Middleware* (2005), G. Alonso, Ed., vol. 3790 of *Lecture Notes in Computer Science*, Springer, pp. 390–400.

[3] BEIGBEDER, T., COUGHLAN, R., LUSHER, C., PLUNKETT, J., AGU, E., AND CLAYPOOL, M. The effects of loss and latency on user performance in unreal tournament 2003. *In the Proceedings of NetGames'04, Portland, Oregon, USA* (August 2004), 144–151.

[4] BESKOW, P., HALVORSEN, P., AND GRIWODZ, C. Latency reduction in massively multi-player online games by partial migration of game state. *Second International Conference on Internet Technologies and Applications, Wrexham, Wales* (September 2007), 153–163.

[5] BROOKES, B. The derivation and application of the Bradford-Zipf distribution. *Journal of Documentation 24*, 4 (1968), 247–265.

[6] CHAMBERS, C., CHANG FENG, W., CHI FENG, W., AND SAHA, D. A geographic redirection service for on-line games. *In Proceedings of the eleventh ACM international conference on Multimedia, Berkeley, CA, USA* (November 2003), 227–230.

[7] CHAMBERS, C., CHANG FENG, W., SAHU, S., AND SAHA, D. Measurement-based characterization of a collection of on-line games. *In the Proceedings of the 5th ACM SIGCOMM Workshop on Internet measurement, Berkeley, CA, USA* (October 2005), 1–14.

[8] CHANG FENG, W., CHANG, F., CHI FENG, W., AND WALPOLE, J. 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, France* (November 2002), 151–156.

[9] CHANG FENG, W., AND CHI FENG, W. On the geographic distribution of on-line game servers and players. *In the Proceedings of NetGames'03, Redwood City, California, USA* (May 2003), 173–179.

[10] CLAYPOOL, M. The effect of latency on user performance in real-time strategy games. *Elsevier Computer Networks 49*, 1 (Sept. 2005), 52–70.

[11] CLAYPOOL, M. Network characteristics for server selection in online games. *In Proceedings of the fifteenth Annual Multimedia Computing and Networking (MMCN'08), San Jose, CA, USA 6818* (January 2008), 681808.

[12] CLAYPOOL, M., AND CLAYPOOL, K. Latency and player actions in online games. *Communications of the ACM 49*, 11 (Nov. 2005), 40–45.

[13] DE VLEESCHAUWER, B., VAN DEN BOSSCHE, B., VERDICKT, T., DE TURCK, F., DHOEDT, B., AND DEMEESTER, P. Dynamic microcell assignment for massively multiplayer online gaming. *In the Proceedings of NetGames' 05, Hawthorne, NY, USA* (Oct. 2005), 1–7.

[14] DICK, M., WELLNITZ, O., AND WOLF, L. Analysis of factors affecting players' performance and perception in multiplayer games. *In the Proceedings of NetGames'05, Hawthorne, NY, USA* (October 2005), 1–7.

[15] DUONG, T., AND ZHOU, S. A dynamic load sharing algorithm for massively multiplayer online games. *In the Proceedings of ICON'03, Sydney, Australia* (October 2003), 131–136.

[16] FUNCOM. Anarchy Online. *http://www.anarchy-online.com/, June* (2008).

[17] GRIWODZ, C., AND HALVORSEN, P. The fun of using TCP for an MMORPG. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* (May 2006), ACM Press, pp. 1–7.

[18] KARAMAN, A., AND HASSANEIN, H. S. Core-selection algorithms in multicast routing - comparative and complexity analysis. *Computer Communications 29*, 8 (2006), 998–1014.

[19] MEDINA, A., LAKHINA, A., MATTA, I., AND BYERS, J. BRITE: Universal topology generation from a user's perspective. Tech. Rep. BUCS-TR-2001-003, Computer Science Department, Boston University, Apr. 2001.

[20] ZNATI, T. B., AND MOLKA, J. A simulation based analysis of naming schemes for distributed systems. In *Proceedings of the 25th Annual Simulation Symposium* (Los Alamitos, CA, USA, Apr. 1992), pp. 42–53.