Dynamic Microcell Assignment for Massively Multiplayer Online Gaming

Bart De Vleeschauwer
bdevlees@intec.ugent.be

Bruno Van Den Bossche
brvdboss@intec.ugent.be

Tom Verdickt
tverdick@intec.ugent.be

Filip De Turck
fdeturck@intec.ugent.be

Bart Dhoedt
dhoedt@intec.ugent.be

Piet Demeester
pietdm@intec.ugent.be

Ghent University - IBBT - IMEC, Department of Information Technology
Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium
Tel: +3293314900, Fax: +3293314899

ABSTRACT

With the number of players of massively multiplayer online games (MMOG) going beyond the millions, there is a need for an efficient way to manage these huge digital worlds. These virtual environments are dynamic and sudden increases in player density in a part of the world have an impact on the load of the server responsible for that section of the virtual world. In this paper we propose the division of the world into several interacting microcells that can be dynamically assigned to a set of servers. We outline the architecture of such a system and describe a set of algorithms that assign the microcells to the available servers. The maximum load experienced by a server is used as a minimization criterion. The different algorithms are compared with each other and with the standard approach used in these games.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems; D.2.11 [**Software Engineering**]: Software Architectures; I.6.5 [**Simulation and Modeling**]: Model Development

General Terms

Algorithms, Performance, Design

Keywords

MMOG, Game Server Architecture, Load Balancing, Microcell Distribution

1. INTRODUCTION

Recent years have seen an enormous increase in the number of players engaging in Massively Multiplayer Online Games (MMOGs). These games offer the player a huge digital environment in which he can interact with other players. One of the more recent incarnations of this game type, World of Warcraft [2], has over 2,000,000 users with peaks of over 500,000 players interacting at the same time in the digital world. Other examples include Second Life [7], a virtual world that even allows users to create their own objects and insert them into the world. With the widespread availability of broadband Internet access and a new generation of game consoles [5, 6, 9] on the horizon, all boasting broadband capabilities, this type of game will no doubt enjoy an even greater popularity in the future.

To support these virtual worlds, with huge numbers of interacting entities, there is a need for an efficient architecture that is able to meet the requirements in terms of generated load. Due to the massive scale of these games, a single machine is not able to support a virtual world with hundreds of thousands of users. As a result all MMOGs distribute the load among a set of servers, a grid or a cluster. The approach used in Second Life is splitting the gameworld into different cells, each managed by a single server. These cells communicate with their neighbours and manage players and objects located within their boundaries.

In this paper we propose to go a step further. A static division of the game world in big cells is not able to react to the dynamic increase in load when events that create local peaks in player density happen. These so called hotspots occur when a location in the game world exhibits a sudden increase in popularity resulting in a high concentration of players in a small part of the world. When using large static cells, it can occur that several hotspots are managed by one single server, resulting in a severe degradation of the game experience when this server can no longer cope with the increased load. We propose splitting the game world into several smaller cells called "microcells". These cells can be dynamically assigned to a set of servers, thus allowing an even distribution of the load in order to minimise the maximum load experienced by one server and to eliminate bottlenecks.

In section 2 the microcell architecture is outlined and the differences with a standard architecture are highlighted. Section 3 contains a description of the algorithms that we have developed to determine a microcell allocation. These algorithms are evaluated in section 4. Finally, in section 5 conclusions are drawn and some directions for future research are described.

2. ARCHITECTURAL OVERVIEW

The overall concept of the game for which an architecture is presented in this paper, follows that of a typical MMOG. Players can move freely through a vast virtual world, interacting with other players and computer-controlled entities.

2.1 Cells

In the architecture presented in this paper, the world is divided into a number of interconnected *cells*. Each cell consists of a part of the world, with its own contents and characteristics. No initial assumptions are made regarding the exact shape of the cells, though in a real-life implementation, the cells will probably have a regular shape, such as square or hexagonal. Also, no assumption is made about the number of cells neighbouring a given cell. Again, in a real-life scenario this might be the same for every cell. Figure 1 depicts a world divided into four cells.
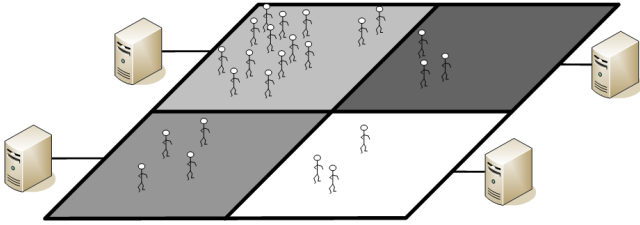

Figure 1: Traditional cell-based world

The underlying goal of this design is to allow the world and all the entities contained in the world to be easily divided over several servers. This concept has already been shown to work in Second Life [7], where the world is also divided into a number of regions, each assigned to a different server, in order to distribute server load.

An important aspect of the design is that the cell concept should be "transparent" to the players. Players should experience a single, vast, unfragmented world. They should be able to seamlessly move from one cell to another, without noticing the cell borders, and players near a cell border should see and be influenced by events in the neighbouring cell. This does of course necessitate the communication between neighbouring cells. Player data should be moved between cells if a player moves from one cell to another, and events and player actions happening near the cell border should be forwarded to the neighbouring cell.

While an architecture with fairly large cells assigned to separate servers (figure 1) distributes the load over several servers, it cannot guarantee an even distribution of the load. The reason is that players will probably not be spread evenly across the virtual world. Some regions, e.g. large cities, will have a much higher player concentration than others, e.g. a desert.

As a result, servers managing the more densely populated cells will have a much higher load than other servers. When there is a very high player density in a certain region, this might cause a situation where one or a few servers can no longer handle their load, leading to unacceptable performance loss in the cells managed by those servers. It would be better to reduce the load in those servers, by shifting part of it to the other, less populated, servers. There is a certain tolerance to performance problems in general and delay specifically: delays below a certain limit do not really influence the game or the user experience [1, 8]. Therefore the slightly higher load in the least-loaded servers will not cause real problems, while the reduced peak loads will prevent bottlenecks from occurring.

2.2 Microcells

One way to solve this problem is to vary the size and shape of the regions managed by a single server. In densely populated areas of the virtual world, smaller regions could be assigned to servers, while larger regions could be used in less populated areas, allowing the load to be spread much more evenly among the servers.

This solution solves the problem if the population density is more or less fixed, which however, is not always the case. A number of events might cause a drastic shift in the population density of certain regions. A huge, week-long festival in a town might suddenly attract players from all over the world, and two player factions waging war over a mountain pass might summon large armies to a previously deserted region.

Clearly, a more flexible division of the world is needed to handle these situations. The research presented in this paper therefore takes the cell concept a step further. Instead of dividing the world into fairly large static cells, each supported by a different server, the world is divided into a significantly larger number of smaller cells that can be dynamically assigned to a set of servers, we call these cells *microcells* (figure 2). A single server now manages not one large cell but a number of microcells. Rebalancing the load between the servers can be performed by moving microcells from a high-load server to a lower-load server, and thereby resizing the regions they are responsible for. This is significantly easier than reshaping or resizing existing cells, and thus is viable even when managing only temporary population density shifts.
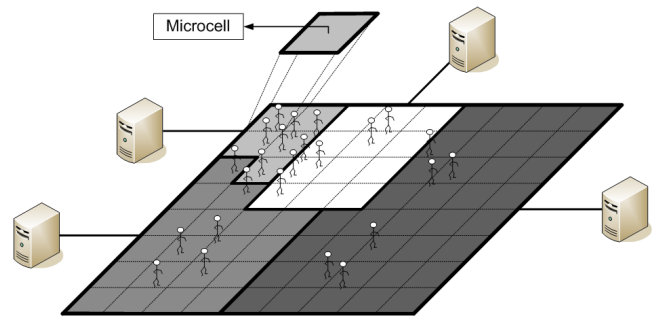

Figure 2: World divided into microcells

Of course, having a larger number of cells causes an increase in the inter-cell communication. While this might increase the average server load, modeling results presented further in the paper indicate that the load of the most heavily-loaded server can be decreased by using microcells,

because of a better spreading of the server load. This decreases the chance of a bottleneck occurring, which is the primary design goal of the platform architecture.

By using a careful server design, the overhead caused by the communication between cells can be somewhat reduced when using microcells. In the "large cell" architecture presented in section 2.1, each cell has its own database, containing among other things the players residing in the cell and the cell's entities. This does not have to be the case in a microcell architecture. Instead, a single database is used for every server, irrespective of the number of microcells present on the server. Apart from the actual cells and the database, each server also contains a controller, managing the cells, the database, and the communication between the cells.
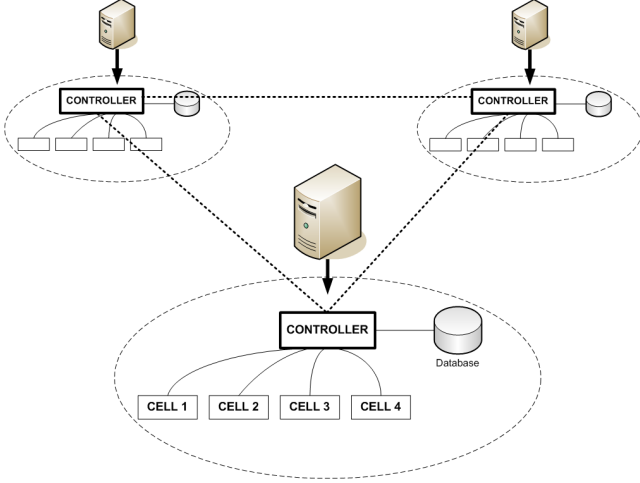

Figure 3: Server architecture

Using the shared database and the controller, communication between microcells on the same server can now be made more efficient. For example, moving a client from one cell to another would usually require the player information to be moved from the database of one cell to that of another. This server architecture, however, allows to avoid such copying for players moving between cells on the same server, since the database is shared between both cells.

While this server design will not completely eliminate the additional overhead of using microcells instead of larger cells, it does allow to reduce that additional overhead significantly, provided that several neighbouring and communicating microcells are supported by the same server. More information on deploying the microcells to the servers will be given in section 3. Together with the possibility, provided by the microcells, to spread the load more evenly across the servers, the maximum load on a server can be limited, with only a moderate increase in average server load. Thus, chances of servers being overloaded are reduced, making bottlenecks and unacceptable game performance occur less frequently.

3. ALGORITHMS

To take full advantage of the microcell approach we need an efficient way to distribute the cells over the servers. This must be done in such a way that the interaction patterns between the microcells are taken into account and that the

maximal load experienced by a server is minimized. The total load experienced by a server is expressed as a weighted sum of all tasks to be performed by that server, these are: processing player actions, forwarding player actions to neighbouring cells, receiving forwarded actions from neighbouring cells, and moving players to and from neighbouring cells. The weights allow to define the load caused by such actions, relative to each other. For example, the movement of a player between cells generates more load than the processing of a simple player action, and a player migration between cells on the same server requires less work than when those cells reside on different servers. In Appendix A a mathematical formulation of the server load function is given. More information on the specific values of the parameters can be found in section 4.

For each microcell the number of actions that are performed within the cell boundaries is specified, as well as the interaction pattern with its neighbouring cells: the forwarding and receiving of actions between cells and the migration of players between cells. Figure 4 shows the information flow of the algorithms. On top there is the logical world, divided into microcells that are specified by their values for actions and migrations. Darker colours indicate a higher player concentration. The world model is handed to the algorithm, together with the load function and the values of the parameters of the algorithm. The result is the allocation scheme that defines to which servers the cells are assigned.
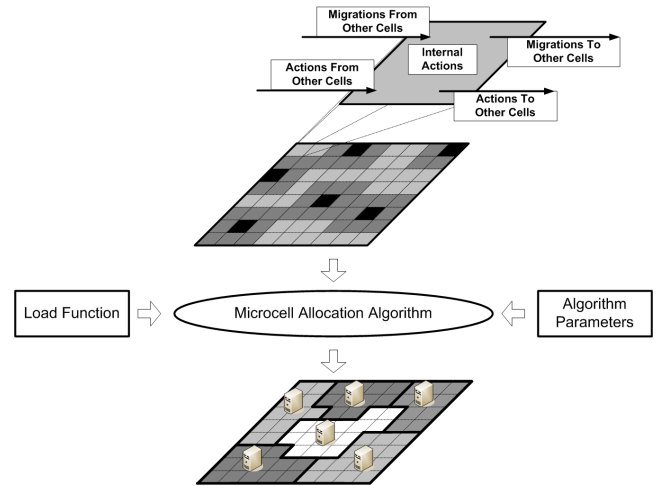

Figure 4: Algorithm Flow Diagram

We developed a set of algorithms that determine the assignment of cells to servers, ranging from a greedy heuristic to an optimal ILP-based (Integer Linear Programming) algorithm.

3.1 Balanced Deployment Algorithm

This is a greedy algorithm, trying to distribute the server load as evenly as possible. Starting with the microcell with the highest number of players and thus the highest load, not taking inter-cell interactions into account, the microcells are added to the servers one by one. The server to add a microcell to is chosen such, that the highest server load is minimized after adding the cell and in case the maximum server load would be equal with the cell being deployed on different servers, the decision will be made based on the

average server load.

While this algorithm will prevent one server being over-loaded, when another server has a much lower load, the algorithm does not explicitly take "locality" into account. It will deploy cells on the servers one by one, without explicitly placing neighbouring cells on the same server. As a result, while the maximum server load might be limited, the average server load is expected to be rather high, compared to the other deployment algorithms.

3.2 Clustering Deployment Algorithm

An algorithm similar to the greedy approach consists of clustering those microcells that will cause the least overhead when placed together on the same server. This algorithm starts by creating a list of clusters, initially each consisting of one microcell. Next it calculates the cost of placing any combination of two clusters on one server and merges the two clusters with the smallest cost when merged. In the following step this process is repeated until the number of clusters equals the number of available servers.

This algorithm has an important disadvantage as in the last few steps it will have to merge clusters with rather large loads, usually resulting in a deployment with at least one highly loaded cluster. An optimization exists in applying this algorithm on a subset of the microcells and adding the remaining microcells one by one to those clusters with the least load.

3.3 Simulated Annealing Deployment Algorithm

Simulated annealing is a technique that can be used to solve combinatorial optimization problems [3]. Starting from an existing solution it gradually tries to find a better one by altering it. The new deployments are generated by randomly swapping microcells between two servers or by moving a microcell from one server to another. If the new deployment is better, the new solution becomes the current solution. Otherwise, the decision to change the current solution is made according to a probabilistic criterion. At first, there is a high chance to change the current solution in one that is worse, but as the algorithm has done more moves, this probability becomes smaller and smaller. Accepting a worse solution is allowed to let the algorithm explore the whole solution space, as a better solution may only be found by first doing an action that degrades the current solution. During our experiments we noticed that this algorithm was able to find good solutions for our problem with random initial deployments, but especially when starting from the solution of one of our other heuristics.

3.4 Optimal Deployment Algorithm

To determine the optimal allocation scheme for a world, we formulated the problem as an Integer Linear Programming (ILP) problem [4]. Solving this formulation using standard techniques allows us to determine the optimal cell allocation scheme. However, due to the low scalability of these solution methods, determining the optimal distribution of even a small world composed of 64 microcells, distributed over 4 servers already spans multiple days on an average desktop computer. Therefore, this approach can only be used for small worlds. Nevertheless, this algorithm provides an interesting benchmark to evaluate how close the solutions of the deployment algorithms are to the optimal solution for small worlds.

By placing an upper bound on the total time the ILP solver searches for a solution, this algorithm can also be used as a deployment heuristic, but does no longer generate an optimal result.

4. EVALUATION

A test framework was designed to simulate and compare the deployment algorithms presented in section 3. The framework applied the algorithms on a given test world to distribute the microcells over a number of servers. The performance of the deployment was then calculated and compared for all applied algorithms.

The world can consist of any number of cells, arranged in an arbitrary fashion. An important characteristic for the world is, for every cell a and cell b, the fraction of the actions in cell a needing to be forwarded to cell b, because they occur close to cell b, and so might influence players in cell b. Another characteristic is the fraction of the players in cell a moving from cell a to cell b during a single time interval. These "migration fractions" also define the steady-state player distribution across the world. Cells with a higher influx of players have a higher player concentration than cells with a high percentage of players leaving the cell.

Though the framework allows world maps with arbitrary shapes, all simulations were performed on square maps, or more accurately toroidal worlds, where a player walking off the top of the world arrives at the bottom, and players walking off the left side arrive back at the right side. We used square cells, where players in a cell could only move to and interact with neighbouring cells. The player distribution was chosen, such that the world contained a number of "two-level hotspots" (see the upper part of figure 4). These hotspots consist of an outer region with a moderately high player concentration, and a centre with a very high player concentration. The rest of the world is assumed to have a much lower player density than the hotspots. The hotspots represent among other things cities, with many players in the city centre and somewhat less players in the city outskirts, but still more than in the rest of the world.

As explained in section 3, the weighted sum of the number of "messages" to be processed by a server was used as a measure of the performance. Important evaluation parameters, therefore, are the weights assigned to the various types of messages, or the various sources of load on the server, like actions by players in a cell on the server, players moving to or from a cell on the same server, etc. An overview of the values used in our simulations is presented in table 1. These parameter values were mostly estimated from measurements on an existing multiplayer game [1].

Table 1: simulation parameters: event weights

	internal	external
player action	1.0	*N/A*
forward action	0.05	0.1
receive forwarded action	0.2	0.4
player emigration	3.0	15.0
player immigration	3.0	15.0

A distinction is made between messages that remain in-

---
[1]http://crossfire.real-time.com

ternal to the server (e.g. player migration between two cells deployed on the same server) and "external" messages, between two servers (player migrations between cells on different servers). The reason is that the interaction between cells on the same server can be made more efficient by using a server architecture as presented in section 2.2.

Player action is the weight given to the processing of a single action, like moving or jumping, performed by a player in the cell, including notifying the players in the same cell of the action and its results. A fraction of those actions needs to be forwarded to a neighbouring cell. The weights for sending and receiving such an action forwarding are presented as *forward action* and *receive forwarded action*, respectively. Player movement between cells has a weight *player emigration* in the cell the player is leaving, and a weight *player immigration* in the destination cell.

To evaluate the performance of the proposed MMOG architecture, several simulations were performed, deploying the cells of the worlds shown in figure 5 over four servers. The grayscale indicates the player densities.
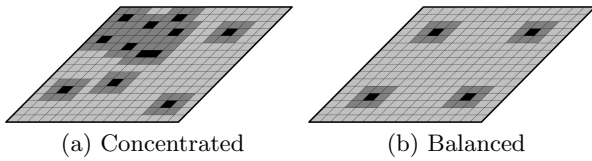

(a) Concentrated   (b) Balanced

Figure 5: A highly concentrated and a perfectly balanced world

It is important to note that all worlds were assumed to contain an equal total number of players. Their only difference lies in the player distribution across the world. In the world shown in figure 5(a), the hotspots are very much clustered together, causing a highly uneven player distribution. Figure 5(b), on the other hand, shows a world containing 4 hotspots, perfectly distributed over the world. Once again, the total number of players is equal for all the worlds, so a higher number of hotspots does not signify a higher number of players, but merely a higher concentration of players in that vicinity.

The figures show the worlds as being divided into 256 microcells (a 16 by 16 grid). The simulations, however, were performed for a varying number of microcells (2 by 2, 4 by 4, 8 by 8, and 16 by 16). In the results we present in this paper, we always used 4 servers to deploy the microcells on. The results for the 2 by 2 world thus represent the traditional large-cell architecture with one cell per server.

The maximum server loads (as defined in section 3 and equation 1) for the map with higher player concentrations (figure 5(a)) and the different deployment algorithms is presented in figure 6. For reference, the maximum server load for the "large-cells" architecture, with a single cell on each server is also shown in the figure.

As can be seen in the figure, a significant performance gain can be obtained by using smaller cells, distributed across the servers, as opposed to a single static cell per server (static in that its size is not dependent on the load or player density in the cell). By dividing each cell into four equal, smaller cells, the maximum server load can be reduced by up to 30%, depending on the deployment algorithm.

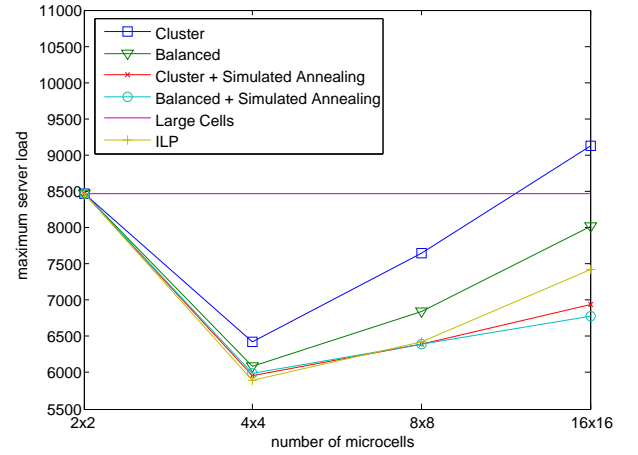Using even smaller cells, however, does not yield an additional performance gain and even increases the maximum

Figure 6: Server load in function of the number of cells for a world with concentrated hotspots

server load. The reason is that the communication overhead between cells increases with an increasing number of cells, offsetting the possible benefit of redistributing the cells over the servers. Therefore, the microcell architecture must be applied with care.

A comparison of the different deployment algorithms shows that by far the best result can be obtained by combining a deployment algorithm with simulated annealing. The choice of algorithm to combine with simulated annealing is less critical, though the balanced algorithm provides a small benefit over the clustered algorithm in this case.

Note that while ILP has been described in section 3.4 as the "optimal deployment algorithm", the results presented in figure 6 are clearly sub-optimal. The reason is that the ILP solver was stopped after an optimization period of two days, resulting in a sub-optimal solution. This shows that ILP, while theoretically providing perfect results, is not usable as a deployment algorithm in this architecture, since the obtained results are worse than those obtained with other algorithms, that required only a few seconds to complete.

The microcell architecture was expected to perform well with high player densities in a small part of the world. Therefore, a similar simulation was performed with a world map with a more even player distribution, as presented in figure 5(b). The maximum server load for the different deployment algorithms is shown in figure 7, the ILP algorithm was again stopped after a period of 48 hours.

In this perfectly symmetrical world, the microcell architecture does not provide a performance gain compared to the large-cell architecture. However, the extra cost of dividing the cells is smaller than 5% for 16 microcells (4 by 4) and 10% for 64 microcells (8 by 8). This is not a large penalty to pay for the increased flexibility. As MMOGs give the users more and more freedom, a more dynamically varying distribution with temporary concentrations will likely be much more typical in future MMOGs.

To evaluate the behaviour of the microcell architecture with different player distributions, we generated two more worlds with the same number of players but with other hotspot distributions. These are depicted in figure 8. These worlds have player concentrations that lie between those of the worlds we used before and allow us to evaluate how
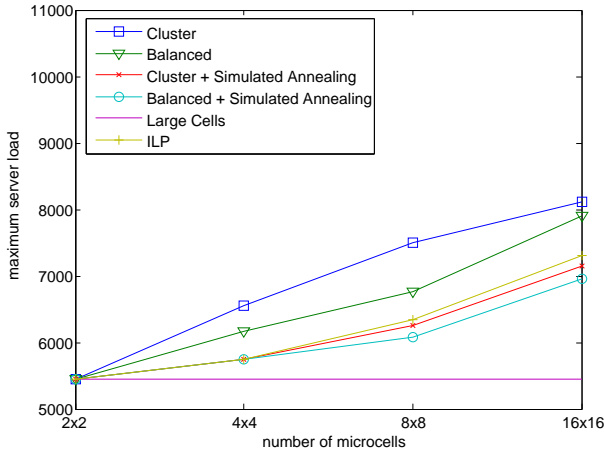
Figure 7: Server load in function of the number of cells for a world with balanced hotspots

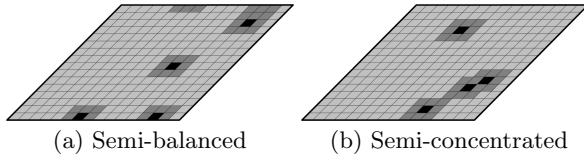(a) Semi-balanced   (b) Semi-concentrated

Figure 8: Additional random worlds

The graph in figure 9 presents the influence of the player distribution on the maximum server load, for the different microcell architectures. The algorithm to determine the microcell allocation was the simulated annealing algorithm applied to the solution generated by the balanced algorithm. When using 4 large cells (and thus, a single cell per server), the maximum load experienced by one server is highly variable, going from just 5500 in a perfectly balanced world to around 8500 in the most concentrated world. The microcell approach however is able to significantly limit the variation of the maximum load. When dividing the world into 16 microcells, the maximum load for the servers, experienced in the different scenarios all lie within 5% of each other. This indicates how well the microcell architecture can dynamically adapt to changing player distributions. A very important consequence of this behaviour is that it is much easier to dimension the servers, as much narrower bounds to the server load can be defined.

Apart from removing peak server load and server bottlenecks and improving flexibility, another goal of this architecture was to spread the load more equally over the servers. The results presented in figure 10 show the maximum and average server load for the Concentrated world model of figure 5. The algorithm to determine the microcell allocation was the simulated annealing algorithm applied to the solution generated by the balanced algorithm. They show a difference of over 35% between the maximum and average server load, when using large cells, which indicates one server is much more loaded than the others. By increasing the number of cells from 4 to 16 this difference is less than 2%. By splitting the world into the small microcells we are
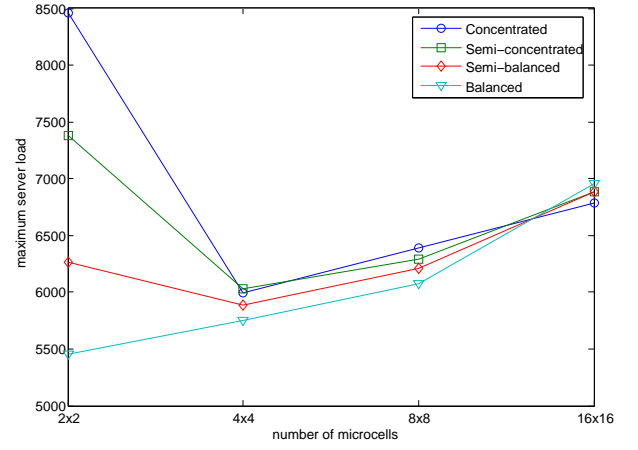

Figure 9: The maximum server load for different player distributions

thus able to evenly distribute the total load generated by the game over the servers.
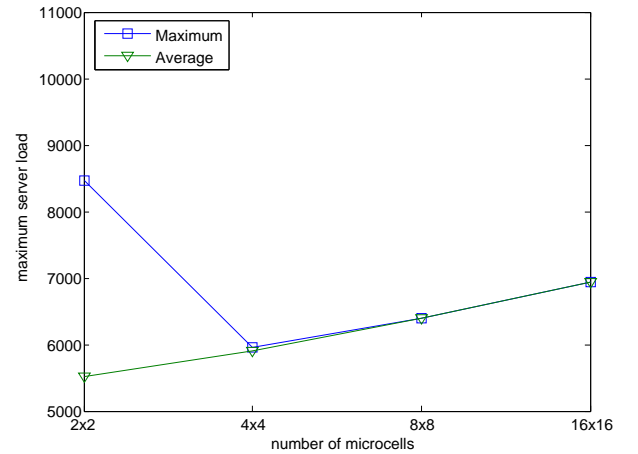

Figure 10: The maximum and average server load of a distribution

To illustrate how a microcell deployment looks, figure 11 shows the cell assignment for the semi-concentrated world (figure 8(b)), divided into 256 microcells, to 4 servers. The algorithm that was used to obtain this result was the simulated annealing algorithm applied to the solution provided by the balanced algorithm. When looking at the figure, one can see the large clustering of cells belonging to the same server. This is to be expected as microcells that belong to the same server will have a lower cell intercommunication overhead.

5. CONCLUSION

This paper presented an MMOG architecture that divides the game world into a number of smaller parts, called microcells, dynamically distributed over the available servers. This is an improvement to similar existing MMOG architectures, where the world is divided into a number of larger cells, each managed by a separate server. We describe a set of algorithms that assign the microcells to the different
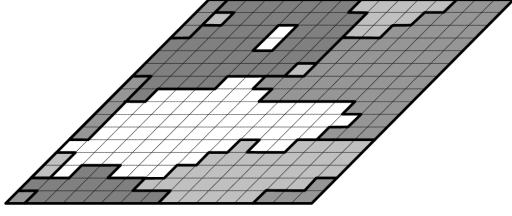
Figure 11: Resulting cell distribution for the semi-concentrated world

servers and use simulations to evaluate the microcell approach.

Simulation results show that a judicious assignment of the microcells to the servers reduces the maximum server load dramatically, compared to large-cell architectures, when the players are not symmetrically distributed over the virtual world. We achieve a load reduction of up to 30%. By minimizing the maximum server load, we greatly decrease the chance of a bottleneck occurring and improve the overall performance perceived by the players. At the same time, both the maximum and the average server load are much less sensitive to the exact player distribution. This greatly facilitates server dimensioning, preventing resource waste.

Future research on this architecture includes a study of its dynamic behaviour. We will focus on how the microcells can be re-assigned to the servers efficiently and study the overall impact on game performance. The automation of this process, in order to automatically re-assign the microcells to the servers in real-time, will also be studied.

6. REFERENCES

[1] G. Armitage. An experimental estimation of latency sensitivity in multiplayer Quake 3. In *Proc. of the 11th IEEE International Conference on Networks (ICON2003)*.

[2] Blizzard Entertainment Press Release. World of warcraft achieves new milestone with two million paying subscribers worldwide. In *[Online] http://www.blizzard.com/press/050614-2million.shtml*.

[3] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[4] G. L. N. Laurence A. Wolsey. *Integer and Combinatorial Optimization*, volume 0-471-82819-X. Wiley-Interscience, 1988.

[5] Microsoft. Xbox 360 fact sheet. In *[online] http://www.xbox.com/en-US/xbox360/factsheet.htm*.

[6] Nintendo press release. Nintendo's compact console will turn the world of gaming on its side. In *[online]http://www.nintendo.com/newsarticle? articleid=02ea1a40-ac09-4cdf-9548-91e5a4e78746&page=other*.

[7] P. Rosedale and C. Ondrejka. Enabling player-created online worlds with grid computing and streaming. *Gamasutra*, September 2003.

[8] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu. The effect of latency on user performance in Warcraft III. In *Proc. of ACM Network and System Support for Games (NetGames)*, pages 3–14, May 2003.

[9] Sony press release. Sony to launch its next generation computer entertainment system. In *http://www.us.playstation.com/Pressreleases .aspx?id=279*.

APPENDIX

A. SERVER LOAD FUNCTION

The mathematical description of the server load is given in equations 1 and 2. Here $serverload_A$ is the load on server A, which consists of the sum of the loads of all cells i that reside on server A. It is assumed that the cells j reside on the same server as cell i, while the cells k reside on a different server (not indicated in the formula, in order to avoid unnecessary clutter). The factor pa represents the player activity, being the average number of actions a player performs during a single time interval. p_i is the number of players in cell i. $ffrac_{ij}$ is the fraction of player actions occurring in cell i that need to be forwarded to cell j, while $mfrac_{ij}$ represents the fraction of the players in cell i that move to cell j in a single time interval. Weight parameters ending in in are parameters for interactions between cells residing on the same server. Weight parameters ending in ex, on the other hand, are used for interactions that are extern to the server (and thus between cells on different servers). The $fsweightxx$ parameters are used for forwarding an action to another cell, while $frweightxx$ is used for receiving such an action. Analogously $emweightxx$ is used for players moving out of the cell, and $imweightxx$ for immigrations (players moving into the cell). For example, $fsweightex$ represents the weight for forwarding an action to a cell on another server, and $imweightin$ the weight for handling a player moving into the cell, from another cell residing on the same server.

$$serverload_A = \sum_i load_i \qquad (1)$$

$$
\begin{aligned}
load_i = {} & pa \times p_i \qquad\qquad\qquad\qquad (2)\\
& + \sum_j pa \times p_i \times ffrac_{ij} \times fsweightin \\
& + \sum_k pa \times p_i \times ffrac_{ik} \times fsweightex \\
& + \sum_j pa \times p_j \times ffrac_{ji} \times frweightin \\
& + \sum_k pa \times p_k \times ffrac_{ki} \times frweightex \\
& + \sum_j p_i \times mfrac_{ij} \times emweightin \\
& + \sum_k p_i \times mfrac_{ik} \times emweightex \\
& + \sum_j p_j \times mfrac_{ji} \times imweightin \\
& + \sum_k p_k \times mfrac_{ki} \times imweightex
\end{aligned}
$$

Note that the servers could be located on a single site, or distributed across the internet. However, the network load was not taken into account in defining the system load function, which makes sense in a localised cluster environment (the most probable server setup in this type of gaming architecture).