Noname manuscript No.
(will be inserted by the editor)

A load balancing scheme for MMOGs

Carlos E. B. Bezerra · Cláudio F. R. Geyer

Received: date / Accepted: date

Abstract MMOGs are applications which require much network bandwidth to function properly. In a distributed MMOG server architecture, with heterogeneous resources, the server nodes may become easily overloaded by the high demand from the players for state updates. In this work, we propose a balancing scheme which has three main goals: cope with the most important load on the servers, which is the network traffic, allocate load on the servers proportionally to the each one's power, and reduce as much as possible the overhead from the distribution. It is is divided in 3 phases: local selection of servers, balancing and refinement. Four algorithms were proposed, from which ProGReGA is the best for overhead reduction and ProGReGA-KF is the most suited for reducing player migrations between servers.

Keywords MMOGs · load balancing · distributed server · graph partitioning

1 Introduction

The main characteristic of massively multiplayer games is the large number of players, having dozens, or even hundreds, of thousands participants simultaneously [1]. This large number of players interacting with one another generates a traffic on the support network which grows quadratically compared to the number of players, in the worst case.

When using a client-server architecture, it is necessary that the server intermediates the communication between each pair of players – assuming that the game is intended to provide guarantees of consistency and resistance to cheating. Obviously, this server will have a large communication load, thus, it must have enough resources (available bandwidth) to meet the demand of the game. We consider here that the main resource to analyse is the available bandwidht, for this is the current bottleneck for MMOGs [2].

Carlos Eduardo Benevides Bezerra, Cláudio Fernando Resin Geyer
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500, Porto Alegre, Brazil
Tel.: +55-51-84065986
Fax.: +55-51-33087308
E-mail: carlos.bezerra@inf.ufrgs.br, geyer@inf.ufrgs.br

The question is, when you use a distributed server, you must delegate to each server node a load proportional to its power. Thus, no matter to which server each player is connected, their game experience will be similar, regarding the response time for their actions and the time it takes to be notified of actions from other players as well as of state changes in the virtual environment of the game.

An initial idea might be to distribute the players between servers, so that the number of players on each server would be proportional to the bandwidth of that server. However, this distribution would not work, for the burden caused by players also depends on how they are interacting with one another. For example, if the avatars of two players are too distant from each other, probably there will be no interaction between them and therefore the server needs only to update each one of them with the result of their own actions. However, if these avatars are close to each other, each player should be updated not only of his own actions, but also of the actions of the other player.

Normally, players can freely move their avatars throughout the game world. This makes possible the formation of *hotspots* [**?**], around which the players are more concentrated than in other regions of the virtual environment. Moreover, many massively multiplayer online RPGs not only permit but also stimulate, to some extent, the formation of these points of interest. In the worlds of these MMORPGs, there are entire cities, where the players meet to chat, exchange virtual goods or even fight, and there are also desertic areas, with few attractions for the players, and where the number of avatars is relatively small compared to other places of the environment.

For this reason, it is not enough just to divide the players between servers, even if this division is proportional to the resources of each one of them. First, in some cases the usage of the server's bandwidth is square to the number of players, while in others it is proportional. That reason alone is enough to get another criterion for load balancing. Moreover, there is another important issue: the existence of hotspots, which motivates the creation of a load balancing scheme for MMOGs to prevent that the presence of hotspots degrade the quality of the game beyond the tolerable.

2 Related work

The servers receive the action performed by a player, calculate its outcome and send it to all interested players, who are usually those whose avatars are close to the avatar of the first player. If two players are split between different servers, each of these need not only to send the state update to the player served by it, but it has also to send the update to the server to which the other player is connected. This server, in turn, forwards that state update to the other player.

It is perceived, then, that each state will be sent twice for each pair of players who communicate through different servers. This overhead not only cause the waste of resources of the servers, but it also increases the delay to update the status of the replica of the game in the players' machines, damaging the interaction between them.

Therefore, players who are interacting with each other should, ideally, be connected to the same server. However, it is possible that all players are linked through relations of interaction. For example, two avatars of two different players, may be distant from each other, but both could be interacting with a third avatar, between the other two. However, it is still necessary to divide them between servers. The question is how

many pairs of players and which of them will be divided into different server nodes. It is therefore necessary to decide a criterion to group players.

Next, some principles, used by other authors, will be presented. Based on some of these principles, and on research and implementation carried out, it will be defined the load balancing scheme proposed here.

2.1 Microcells and macrocells

One way to use the locality of the players is grouping them according to the position occupied by their avatars in the virtual environment. The question would be how to form such groups. One way of doing this would be dividing the world of the game in several cells connected to one another. Each cell would be a part of the world, with its own content and characteristics, which would be assigned to a server node. The easiest way to do this is with a grid of cells of the same size and shape.

An important aspect of this design is that the concept of cell is transparent to the players. They see a wide, single and contiguous world, even if they are repeatedly crossing the boundaries between different cells. Obviously, this requires that the cells communicate, updating and notifying each other of events that occurred near the border between them, and of the migration of players from one to the other.

Although this approach with cells distribute the load among several servers, there are no guarantees that this distribution will be uniform, because of the high mobility of players and the existence of hotspots. One of the ideas proposed in the literature [3] also follows the principle of dividing the virtual environment in cells of fixed size and position, but these cells are relatively small – or **microcells** – and they can be grouped, forming an area called **macrocell**. Each macrocell is then assigned to a different server, which will manage not a large cell of fixed size and position, but a variable set of small cells. These microcells can then be moved dynamically between different macrocells, maintaining the load on each one of the servers under a tolerable limit.

Obviously, the microcells designated to the same server node do not generate additional traffic to synchronize with each other, but the synchronization overhead of the macrocell is unpredictable, because the number of neighbors of each one of them is not foreknown, for its shape is variable. However, it was demonstrated [3] that this overhead is compensated by a better distribution of the load between servers in the game.

2.2 Load balancing in local scope

In [4], it is also proposed a scheme for dynamic load balancing for the servers of a multi-server virtual environment, taking into consideration that users can be distributed through this world in a non-uniform way. Following the scheme proposed by the authors, an overloaded server starts the process by selecting a number of other servers to be part of the load redistribution. The set of selected servers depend on the load level of the initiating server as well as on the amount of idle resources of the other servers. After the formation of this set, its elements allocate portions of the virtual environment using a graph partitioning algorithm, so that the servers involved have similar final load.

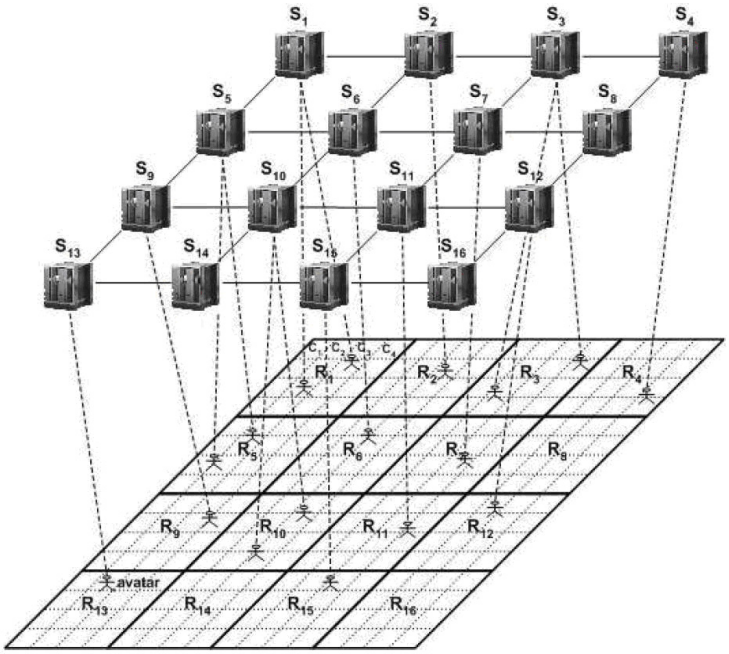

Fig. 1 Cells and regions in a distributed server system

The main aspect of the solution proposed by the authors was the use of local information (the server that initiated the balancing process and its neighbors), rather than global information (involving all servers in the balance). The former shows small overhead, but it can not solve the problem efficiently in a few steps, as overloaded servers tend to be adjacent. The global approach, in turn, is able to divide the workload in the most balanced way possible, but its complexity grows exponentially with the number of servers involved. The solution is then by involving only a subset of servers, such that its cardinality varies according to the need (if the neighbors of the server which triggered the load balancing are also overburdened, more servers are selected) . Thus, there is a little more information than the local approach, but without the complexity inherent to the global one.

To tackle the problem, the authors also subdivide the virtual environment in rectangular **cells** – similar to the microcrells. The cells are grouped in **regions** – or macrocells – and each region is managed by one server. Each server keeps its players updated of the state of the virtual environment, while also handling the interactions between avatars in its region. Two cells are adjacent (or neighbors) if they share a border. Similarly, two regions – and their servers – are adjacent if there is a pair of adjacent cells, each of which belong to one of the two regions. Figure 1 illustrates a distributed server system, consisting of 16 nodes. The virtual environment is divided in 256 cells, which are grouped in 16 regions.

The workload of a cell was defined as the number of avatars present in that cell. The authors assumed that all players receive state updates in the same frequency, so that the burden of processing (computing and communication) that a cell requires from a server is proportional to the number of users in that cell. The workload of

a region and its designated server is defined as the sum of the individual workloads of the cells which form the region. Each server periodically evaluates its workload and exchange this information with its neighbors. They have assumed, too, that these servers are connected through a high-speed network. Thus, the overhead to exchange workload information among neighbors is limited and considered negligible compared to other costs of the distribution. For the same reason, they also assumed as negligible the overhead of communication between servers in different regions when players are interacting.

2.2.1 Selection of local server group to balance the load

A server starts the balancing when the load assigned to it is beyond its power. This server selects a number of other servers to get involved with the distribution. First, it chooses the least loaded server among its neighbors and sends a request that he participates in the load balancing. The chosen server rejects the request if it is already involved in another balancing group, otherwise it responds to the server with the load information of its own neighbors. If the selected neighbor server is unable to absorb the extra workload of the initiating server, the selection is performed again among the neighbors not only of the overloaded server, but also the neighbors of the neighbor chosen in the first step. The selection continues until the workload of the first server can be absorbed – that is, the workload of all selected servers becomes smaller than a certain limit.

Figure 2 illustrates the operation of the algorithm. All servers have the same capacity, each one being able to handle 100 users. First, the initiator server, S_6 is inserted into SELECTED and its neighbors (S_2, S_5, S_7 and S_{10}) are added to CANDIDATES (Figure 2(a)). So S_7, which has the lowest workload among the servers in CANDIDATES, is selected and invited to participate in the load distribution. When S_7 sends to S_6 the workload information of its neighbors (S_3, S_6, S_8 and S_{11}), S_7 is inserted into SELECTED and its neighbors, except S_6, are added to CANDIDATES (Figure 2(b)). Now, S_{11}, which has the lowest workload among servers in CANDIDATES, is selected and invited to participate in the load distribution. However, S_{11} rejects the invitation, because it is involved in another distribution, initiated by S_{12}. Thus, S_{11} is removed from CANDIDATES and S_{10} is selected because it now has the lowest workload among all servers in CANDIDATES (Figure 2(c)). This process continues until the average workload is under a pre-defined threshold (Figure 2(d) and Figure 2(e)).

Para exemplificar o funcionamento do algoritmo, pode-se observar a Figura 2. Todos os servidores têm a mesma capacidade, podendo cada um comportar 100 usuários. Primeiro, o servidor iniciador, S_6, é inserido em SELECIONADOS e seus vizinhos (S_2, S_5, S_7 e S_{10}) são adicionados a CANDIDATOS (Figura 2(a)). Então, S_7, que tem a menor carga de trabalho dentre os servidore em CANDIDATOS, é selecionado e convidado a participar da distribuição de carga. Quando S_7 responde a S_6 com a informação de carga de seus vizinhos (S_3, S_6, S_8 e S_{11}), S_7 é inserido em SELECIONADOS e seus vizinhos, exceto S_6, são adicionados a CANDIDATOS (Figura 2(b)). Agora, S_{11}, que tem a menor carga de trabalho dentre os servidores em CANDIDATOS, é selecionado e convidado a participar da distribuição de carga. Porém, S_{11} rejeita o convite, pois já está envolvido em outra distribuição, iniciada por S_{12}. Assim, S_{11} é removido de CANDIDATOS e S_{10} é selecionado porque tem agora a menor carga de trabalho dentre os servidores em CANDIDATOS (Figura 2(c)). Até que a carga de trabalho média

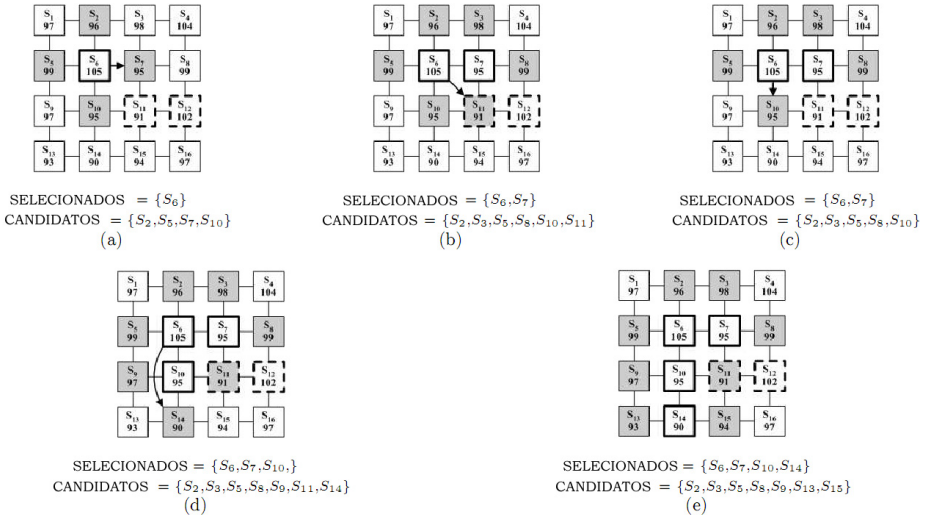

Fig. 2 Selecting the group of servers for local rebalance

dos servidores em SELECIONADOS se torne menor que $0,9 \times CP$, ou seja, 90, o procedimento acima continua (Figura 2(d) e Figura 2(e)).

After finishing the local server set, the initiating server performs a load rebalancing in such group, using some graph partitioning algorithm. To map the virtual environment to a graph, each cell is represented by a vertex, whose weight equals to the number of avatars in that cell; and every two vertices which represent adjacent cells are connected by an edge.

3 Proposed load balancing scheme

In the previous section it was presented some existing works regarding load balancing in MMOGs when using multiple servers to provide the support network. The load balancing scheme proposed in this work is based on some of the principles in the literature. One of them is the division of the virtual environment in microcells, for later grouping in macrocells. This is a relatively simple way of addressing the issue of the avatars' movement dynamics through the game world, by transfering the microcells dynamically according to need.

It will also be used the idea of balancing based only on local information – each server, when needing to reduce its workload, selects only a few other servers to join a local load rebalancing. Thus, we can greatly reduce the complexity of balancing because it will not be necessary that all servers in the game exchange messages among themselves every time that any one of them is overloaded.

The related works, as described in the previous section, try to solve the problem of dynamic load balancing. However, several improvements need to be made in order to make them more coherent with the needs of MMOGs. For example, it was ignored that, usually, the traffic generated by the players is not simply linear, but square for each cluster of players. This misunderstanding can generate considerable differences between the actual load of each server and load estimated by the balancing algorithm.

Another issue that was left out is the overhead, both in the delay of sending messages, as in the use of bandwidth of the servers, where players connected to different servers are interacting with each other. In [4], it is considered that the servers are all in the same high-speed and low latency local-area network, and therefore overhead is negligible. However, when considering a geographically distributed server system, this assumption cannot be made. This overhead must be taken into account, no matter which load balancing algorithm is being used.

One more important point that was not properly considered by previous works, is that the main criterion to be considered when balancing the of MMOG servers is the bandwidth, and not much processing power. Several games include simulations of virtual environments of hundreds or thousands of entities, such as the game Age of Empires [**?**], which are performed without problems in personal computers today. However, if this game were multiplayer and each of these entities were controlled by a player in a network, connected to other players, it would most likely generate a traffic ammount which would hardly be supported by a domestic connection [2].

Moreover, the upload bandwidth must be taken into account, much more than download. This happens for two reasons: first, the usage of the download bandwidht of each server node grows linearly with the number of players connected to it, while the usage of the upload bandwidth may have a quadratic growth, getting quickly overwhelmed; second, the domestic connections – which are majority in a volunteer peer-to-peer system – usually have an upload bandwidth much smaller than the download one.

3.1 Definitions

We also use the idea of mapping the virtual environment on a graph, which will then be partitioned to distribute the workload of the game between the different servers. It is necessary first to define some terms which will be used on the proposed algorithms.

- **Server**: here, server is defined as a node belonging to the distributed system to serve the game. Each server can be assigned a single region;
- **Server power**: the server power, $p(S)$ is a numerical value proportional to the server's upload bandwidth;
- **Server power fraction**: given a set of servers $Servers = \{S_1, S_2, ..., S_n\}$, the power fraction of a server S, $frac_p(S)$, is equal to its power divided by the summed power of all servers in $Servers$:

$$frac_p(S) = \frac{p(S)}{\sum_{i=1}^{n} p(S_i)}$$

- **System power**: the total power of the system, P_{total}, equals the sum of the powers of the n servers which form it:

$$P_{total} = \sum_{i=1}^{n} p(Si)$$

- **Cell**: similar to the microcells, it is considered here the environment being divided into small cells, each one with fixed size and position. If two cells share a border, they are said to be **adjacent** or **neighbors**;

- **Region**: the cells are grouped, forming what is called regions. Usually these areas are contiguous, although in some cases the subgraph that represents them may be disconnected, resulting in the presence of cells isolated from each other. Each region is assigned to a server, and only one, and $s(R)$ is the server associated to the region R. It may be refered throughout the text to region's "power", which in fact refers to the power of the server associated with that region, i.e. $p(s(R))$;
- **Relevance**: the relevance of an avatar A_i to another one, A_j, determines the frequency of updates of the state of A_i the server should send to the player controlling A_j [5]. It may be represented by the function $R(A_i, A_j)$;
- **Avatar's weight**: to each avatar there are various other entities (here, we only consider avatars) of the game, each with a frequency of state updates that need to be sent to the player who controls that avatar. Thus, for each avatar A , its individual weight – or of upload bandwidth that the server uses to send state updates to its player – $w_a(A)$ depends on which other entities are relevant to it, and how much. Let $\{A_1, A_2, ..., A_t\}$ be the set of all avatars in the virtual environment, we have:

$$w_a(A) = \sum_{i=1}^{t} R(A, Ai)$$

- **Cell's weight**: here, the total weight of a cell (or the use of upload bandwidth of its server) will be equal to the sum of the individual weights of the avatars in it. Consider cell C, where the avatars $\{A_1, A_2, ..., A_n\}$ are present. Also, consider that the avatars $\{A_1, A_2, ..., A_t\}$ are all the avatars in the virtual environment. The weight of this cell, $w_c(C)$, is:

$$w_c(C) = \sum_{i=1}^{n} w(Ai) = \sum_{i=1}^{n} \sum_{j=1}^{t} R(A_i, A_j)$$

- **Region's weight**: the weight of a region is the sum of the weights of the cells that compose it. Let R be the region formed by $\{C_1, C_2, ..., C_p\}$. The weight of R is:

$$w_r(R) = \sum_{i=1}^{p} w_c(Ci)$$

- **Region's weight fraction**: given a set of regions $Regions = \{R_1, R_2, ..., R_n\}$, the weight fraction of R, relative to $Regions$, is:

$$frac_r(R) = \frac{w_r(R)}{\sum_{i=1}^{n} w_r(R_i)}$$

- **Region's resource usage**: fraction that indicates how much of the power of the server of that region is being used. It is defined by:

$$u(s(R)) = \frac{w_r(R)}{p(s(R))}$$

- **World weight**: the total weight of the game, $W_(total)$, depends on how the distribution will be made. It will be used as a parameter for the partitioning of the virtual environment. It is the sum of the weight of all cells. Let $\{C_1, C_2, ..., C_w\}$ be the set of all cells in which the game world is divided, we have:

$$W_{total} = \sum_{i=1}^{w} w_c(Ci)$$

– **System usage**: fraction that indicates how much of system resources as a whole is being used. It is defined by:

$$U_{total} = \frac{W_{total}}{P_{total}}$$

– **Cell interaction**: the interaction between two cells is equal to the sum of all interactions between pairs of avatars where each one of them is located in one of these cells. The interaction between cells C_i and C_j is given by:

$$Int_c(C_i, C_j) = \sum_{i=1}^{m} \sum_{j=1}^{n} R(A_i, A_j),$$
$$\text{where } A_i \text{ is in } C_i \text{ and } A_j \text{ is in } C_j.$$

– **Overhead between two regions**: if there is only a server and a region, comprising the entire virtual environment of the game, the use of upload bandwidth of the server will be proportional to the W_{total}. However, due to its distribution among various servers, there is the problem of having players from different regions interacting with each other very close to the border between the regions. Because of that, each state update of these players' avatars will be sent twice. For example, let A_i be the avatar of the player P_i, connected to the server S_i, and A_j the avatar of the player P_j, connected to the server S_j. In order for to P_i interacts with P_j, it is necessary that S_i send the state of A_i to S_j, which then forwards to P_j. The same happens in the other way around. The overhead between regions R_i and R_j is equal, therefore, to the sum of interactions between pairs of cells where each one of them is in one of these regions. If R_i and R_j have respectively m and n cells, we have that the interaction – or *overhead* – between them is given by:

$$Int_r(R_i, R_j) = \sum_{i=1}^{m} \sum_{j=1}^{n} Int_c(C_i, C_j),$$
$$\text{where } C_i \in R_i \text{ e } C_j \in R_j.$$

– **Total Overhead**: the total overhead on the server system is calculated as the sum of overheads between each pair of regions. So we have:

$$OverHead = \sum_{i} \sum_{j,j \neq i} Int_r(Ri, Rj)$$

As the concepts needed to understand the proposed load balancing scheme have been defined, now it will be described how the virtual environment is mapped on a weighted graph, which will then be partitioned. Let $GW = (V, E)$ be a graph that represents the game world, where V is the set of vertices and E is the set of edges connecting vertices. Each component of this graph, and what it represents, is described below:

– **Vertex**: each vertex in the graph represents a cell in the virtual environment;
– **Edge**: each edge in the graph connects two vertices that represent adjacent cells;
– **Partition**: each partition of the graph GW – a subset of the vertices of GW, plus the edges that connect them – represents a region;

Fig. 3 Mapping of the virtual environment on a graph

- **Vertex weight**: the weight of each vertex is equal to the weight of the cell that it represents;
- **Edge weight**: the weight of the edge connecting two vertices is equal to the interaction between the cells represented by them;
- **Partition weight**: the weight of a partition is equal to the sum of the weights of its vertices, i.e. the weight of the region that it represents;
- **Edge-cut**: the edge-cut in a partitioning is equal to the sum of the weights of all the edges which connect vertices from different partitions. This value is equal to the sum of the overheads between each pair of regions. Thus, the edge-cut of the graph GW is equal to the total overhead on the server system.

Figure 3(a) illustrates how the mapping is done with square cells, while figure 3(b) shows how it would be with hexagonal cells. The objective of the balancing scheme proposed here is to assign to each server a weight proportional to its capacity, reducing as much as possible the edge-cut of the graph that represents the virtual environment and, consequently, reduce the overhead inherent to the distribution of the game on multiple servers. Although this is an NP-complete problem [**?**], efficient heuristics will be used to reduce this overhead. In the following sections the algorithms proposed in this work will be presented.

3.2 Proposed algorithms

It is considered that an initial division of the virtual environment has already been made. Each server should then check regularly if there is an imbalance and trigger the algorithm. While the overhead resulting from the distribution of virtual environment is part of the workload on servers, there is no way to know it beforehand without executing the repartitioning first. For this reason, the "weight" to be distributed does not include this extra overhead.

When there is an unbalanced region, it is selected a local group of regions, similar to what was shown in section 2.2, but with some changes (section 3.2.1). After this selection, it will be used an algorithm whose parameters are only the loads of the cells and their interactions, because such data is available. Finally, with the regions already balanced, they will be mapped on a graph and the algorithm of Kernighan and Lin [6] will be used to refine the partitioning, reducing the edge-cut and, thus, the overhead, while keeping the balance.

The proposed scheme is then divided into three phases:

1. Select the group of local regions;
2. Balance these regions, assigning to each one a weight which is proportional to the power of its server;
3. Refine the partitioning, reducing the overhead.

A decision to this balancing scheme is that a region R is considered overloaded only when the use of its resources is greater than the total use of the system, also considering a certain tolerance, tol, to avoid constant rebalancing. Thus, the server starts the balancing of R when, and only when, $u(s(R)) > U_{total} \times tol$. Thus, even

if the system as a whole is overburdened, a similar quality of game will be observed among the different regions, dividing the excessive weight between all servers fairly. What can be done when $U_{total} > 1$ is gradually reduce the amount of information sent in each state update, leaving for the application the task of extrapolating the missing information based on previous updates.

Another important aspect is that each server always has an associated region. What might happen would be a region being empty – without any cells – and its server not participating in the game. This is useful when the total capacity of the server system is much greater than the total load of the game, or $P_{total} \gg W_{total}$. In this case, the introduction of more servers would only increase the communication overhead of the system, without improving its quality – except when introduced to provide fault tolerance.

The algorithms were developed oriented to regions, instead of servers, in order to be more legible, becaus of the constant transfers of cells. Moreover, it becomes easier to extend, in the future, the balancing model used here to allow more than one server managing the same region. The proposed algorithms are described as follows. The one in section 3.2.1 is for phase 1, the sections 3.2.2 to 3.2.5 are alternatives for phase 2 and the algorithm in section 3.2.6 is the refinement of phase 3.

3.2.1 Local regions selection

The algorithm for selection of regions (Algorithm 1) aims to form a set of regions such that the average usage of resources of the servers of these regions is below a certain limit. Starting from the server that has started the balancing, its neighboring regions with the least usage of resources are added. When the average usage is less than 1, or less than U_{total} (line 5), the selection ends and phase 2 begins, with the region set as input. These two conditions are justified because there are two possibilities: $U_{total} \leq 1$ and $U_{total} > 1$.

In the case when $U_{total} \leq 1$, there is sufficient power in the system so that all servers have a usage smaller than 100%. Thus, regions are added to the group until all the servers involved are using fewer resources than they have. However, when $U_{total} > 1$, there is no way to all servers be using less than 100% of its resources at the same time. Thus, it is sufficient that all servers are similarly overburdened, and that some kind of adjustment is made, which will probably be a reduction in the information sent to players in each state update.

If even after all the neighbors, and the neighbors of the neighbors and so on, are selected, the criterion is not met, empty regions – belonging to idle servers – will be inserted in the group (line 8) because the overhead of interaction between regions introduced by them is justified by the need for more resources.

3.2.2 ProGReGA

ProGReGA, or proportional greedy region growing algorithm, seeks to allocate the heaviest cells to the regions managed by the most powerful servers. As input, the algorithm receives a list of the regions to balance. Details are shown in Algorithm 2.

Like we said, it is passed as input a list of the regions whose load will be rebalanced. This makes possible the use of this algorithm both in local and global scope, just by choosing between passing some regions or all regions of the environment. The

Algorithm 1 Local regions selection

1: $local\_group \leftarrow \{R\}$
2: $local\_weight \leftarrow w_r(R)$
3: $local\_capacity \leftarrow p(s(R))$
4: $average\_usage \leftarrow \frac{local\_weight}{local\_capacity}$
5: **while** $average\_usage > max(1, U_{total})$ **do**
6: **if** there is any not selected region neighbor to one of $local\_group$ **then**
7: $R \leftarrow$ not selected region neighbor to one of $local\_group$, with smallest $u(s(R))$
8: **else if** there is any empty region **then**
9: $R \leftarrow$ empty region with highest $p(s(R))$
10: **else**
11: stop. no more regions to select.
12: **end if**
13: $local\_weight \leftarrow local\_weight + w_r(R)$
14: $local\_capacity \leftarrow local\_capacity + p(s(R))$
15: $average\_usage \leftarrow \frac{local\_weight}{local\_capacity}$
16: $local\_group \leftarrow local\_group \cup \{R\}$
17: **end while**
18: run phase 2 passing $local\_group$ as input.

Algorithm 2 ProGReGA

1: $weight\_to\_divide \leftarrow 0$
2: $free\_capacity \leftarrow 0$
3: **for** each region R in $region\_list$ **do**
4: $weight\_to\_divide \leftarrow weight\_to\_divide + w_r(R)$
5: $free\_capacity \leftarrow free\_capacity + p(s(R))$
6: temporarily free all cells from R
7: **end for**
8: sort $region\_list$ in decreasing $p(s(R))$ order
9: **for** each region R in $region\_list$ **do**
10: $weight\_share \leftarrow weight\_to\_divide \times \frac{p(s(R))}{free\_capacity}$
11: **while** $w_r(R) < weight\_share$ **do**
12: **if** there is any cell from R neighboring a free cell **then**
13: $R \leftarrow R \cup \{$neighbor free cell with the highest $Int_c(C)\}$
14: **else if** there is any free cell **then**
15: $R \leftarrow R \cup \{$heaviest free cell$\}$
16: **else**
17: stop. no more free cells.
18: **end if**
19: **end while**
20: **end for**

distribution is based on information from that set of regions, whose total weight and total power are calculated in lines 1 to 7. To be redistributed later, all cells associated with these regions are released (line 6).

To provide a partitioning which is balanced, proportional and with low edge-cut since the second phase of the balancing, the regions are sorted in decreasing order of server power (line 10). The ProGReGA then runs through this list, seeking to assign heavier cells to more powerful servers.

In line 12, it is calculated the weight share for each region, considering the total weight that is being divided and the total power of the servers of those regions. The server power fraction of a region, $\frac{p(s(R))}{free\_capacity}$ should be the same fraction of weight that must be attributed to it. Even if this weight is greater than the power of that

server, resulting in an overload, all the servers are similarly overloaded, satisfying the criterion of balance that has been defined. The condition for the end of the allocation of new cells in a region is that its weight is greater than or equal to its weight share.

The choice of cells to include in the region tries to put the most interacting pairs of cells in the same regions, reducing the overhead.

In the first step of the cycle starting in line 16, if the region does not yet have any cell, ProGReGA gets the heaviest free cell (line 20). The same occurs when a region is compressed between the borders of other regions and has no free neighbor, getting cells from somewhere else. This may generate fragmented regions and possibly increase the overhead of the game. However, this happens more often in the last steps of the distribution, when most of the cells would be already allocated to some region. Because the algorithm is greedy, when it reaches that stage of its execution, the free cells would probably the lightest cells of the environment, causing little overhead.

3.2.3 ProGReGA-KH

A possible undesirable effect of the ProGReGA algorithm is that by releasing all the cells and redistributing them, it might happen that one or more regions completely change their place, causing several players to disconnect from their servers and reconnect to a new one. To try to reduce the likelihood of such event, we propose a variation of the algoritgh, called, **ProGReGA-KH** (proportional greedy region growing algorithm keeping heaviest cell). This new algorithm (shown in Algorithm 3) is similar to the original version, except that each region maintains its heaviest cell (lines 6 and 8), from which a region similar to the previous one can be formed, so that several players will not need to migrate to other server. However, to keep one of the cells of each region, it might be preventing a better balancing to occur. Also, the fixation of that cell might cause fragmented regions, increasing the overhead.

Algorithm 3 ProGReGA-KH

```
 1: weight_to_divide ← 0
 2: free_capacity ← 0
 3: for each R in region_list do
 4:     weight_to_divide ← weight_to_divide + w_r(R)
 5:     free_capacity ← free_capacity + p(s(R))
 6:     c ← heaviest cell from R
 7:     temporarily free all cells from R
 8:     R ← R ∪ {c}
 9: end for
10: sort region_list in decreasing p(s(R)) order
11: for each region R in region_list do
12:     weight_share ← weight_to_divide × p(s(R)) / free_capacity
13:     while w_r(R) < weight_share do
14:         if there is any cell from R neighboring a free cell then
15:             R ← R ∪ {neighbor free cell with the highest Int_c(C)}
16:         else if there is any free cell then
17:             R ← R ∪ {heaviest free cell}
18:         else
19:             stop. no more free cells.
20:         end if
21:     end while
22: end for
```

3.2.4 ProGReGA-KF

Another way of trying to minimize the migration of players between servers because of the rebalancing is the **ProGReGA-KF**, or proportional greedy region growing algorithm keeping usage fraction (Algorithm 4). In this algorithm, each region will gradatively release its cells in increasing order of weight, until its weight fraction is less than or equal to the power fraction of its server. Thus, the heaviest cells remain on the same server and, therefore, most players do not need to migrate. After that, the cells that were released are redistributed among the regions with lowest server resource usage (line 13). The disadvantage of this algorithm is, as in ProGReGA-KH the possibility of fragmenting the regions, with many isolated cells, increasing the overhead.

Algorithm 4 ProGReGA-KF

```
 1: weight_to_divide ← 0
 2: free_capacity ← 0
 3: for each region R in region_list do
 4:    weight_to_divide ← weight_to_divide + w_r(R)
 5:    free_capacity ← free_capacity + p(s(R))
 6:    cell_list ← list of cells from R in increasing order of weight
 7:    while frac_r(R) > frac_p(s(R)) do
 8:       C ← first element from cell_list
 9:       remove C from R
10:       remove C from cell_list
11:    end while
12: end for
13: sort region_list in increasing order of u(s(R))
14: for each region R in region_list do
15:    weight_share ← weight_to_divide × p(s(R)) / free_capacity
16:    while w_r(R) < weight_share do
17:       if there is any cell from R neighboring a free cell then
18:          R ← R ∪ {neighbor free cell with the highest Int_c(C)}
19:       else if there is any free cell then
20:          R ← R ∪ {heaviest free cell}
21:       else
22:          stop. no more free cells.
23:       end if
24:    end while
25: end for
```

3.2.5 BFBCT

The **BFBCT** (best-fit based cell transference) is proposed here as an alternative to ProGReGA and its variants. The objective of the algorithm is to check what is the weight excess in each region and transfer it to free regions whose capacity is the closest to that value. This is done by transferring cells whose weight is the closest to free capacity of the receiving region, observed two restrictions: first, the total weight transferred can not be larger than the free capacity of the destination region and, second, we should not transfer a load greater than the necessary to eliminate the overload. The second restriction is justified because a weight transfer larger than necessary would probably result in a larger amount of migrating players. The exception to this rule is when a cell is heavier than the other, not for having more avatars, but because they

are closer, with quadratic traffic growth between them. The Algorithm 5 describes in detail the operation of BFBCT.

Algorithm 5 BFBCT

```
1: for each region R_i in region_list do
2:    weight_to_lose ← w_r(R_i) − W_total × frac_p(s(R_i))
3:    destination_regions ← region_list − {R_i}
4:    sort destination_regions in decreasing order of u(s(R))
5:    for each region R_j in destination_regions do
6:       free_capacity ← frac_p(s(R_j)) × W_total − w_r(R_j)
7:       weight_to_this_region ← min(weight_to_lose, free_capacity)
8:       while weight_to_this_region > 0 do
9:          if R_i has a cell C such that w_c(C) ≤ weight_to_this_region then
10:            C ← cell from R_i with weight closest to, but not larger than,
               weight_to_this_region
11:            R_i ← R_i − {C}
12:            R_j ← R_j ∪ {C}
13:            weight_to_this_region ← weight_to_this_region − w_c(C)
14:            weight_to_lose ← weight_to_lose − w_c(C)
15:         else
16:            continue with next R_j.
17:         end if
18:      end while
19:   end for
20: end for
```

3.2.6 Refining with the KernighanLin algorithm

After balancing the load between servers in phase 2, each server will have a usage ratio similar to the others. However, phase 2 does not consider the interaction between cells and the overhead between regions, which may have been fragmented, resulting in many pairs of neighboring cells that are not part of the same region. This increases the probability that two players, each in one of these cells, interact with each other close to the border, which, as was shown in previous sections, causes a waste of resources of the server system.

To minimize this problem, it is proposed here to use the algorithm of Kernighan and Lin [6]. Although the algorithm receives as input a graph, the vertices, edges, weights and partitions can be interpreted respectively as cells, interactions, loads and regions. Given a pair of regions, KernighanLin searches for pairs of cells that, when exchanged between their regions, the overhead is reduced.

Let R_A R_B be two regions and a a cell such that $a \in R_A$. The external interaction of a is defined as $E(a) = \sum_{b \in R_b} Int_c(a, b)$ and the internal interaction is defined as $I(a) = \sum_{a' \in R_A} Int_c(a, a')$. The cell a is prone to change regions if its interaction with R_B is greater than its interaction with R_A, i.e. $D(a) = E(a) - I(a) > 0$. Assuming that there is a cell $b \in R_B$ such that $D(a) + D(b) - 2 \times Int_c(a, b) > 0$, the exchange will reduce the overhead between the regions. The term $D(a)+D(b)-2\times Int_c(a, b)$ is refered as $gain(a, b)$, because it represents the gain (reduction of overhead) of exchanging the regions of a and b.

In the case of a virtual environment distributed among various regions, generally more than two, KernighanLin is run for each pair of regions. Moreover, after each

exchange the balance should be ketp – otherwise, all cells could go to the same region, completely eliminating the overhead. The KernighanLin is widely known in the area of distributed systems, so detail will not be provided here. Consider only that it returns a value of true if any change was made and false if it not. It runs for all pairs of regions. until it returns a value of false, indicating that no exchange will provide additional gain. The Algorithm 6 shows how the algorithm KernighanLin is called.

Algorithm 6 KernighanLin

```
1: swapped ← true
2: while swapped = true do
3:    swapped ← false
4:    for each region R_i in region_list do
5:       for each region R_j in region_list do
6:          if KernighanLin(R_i, R_j) = true then
7:             swapped ← true
8:          end if
9:       end for
10:   end for
11: end while
```

4 Simulations and results

To perform the simulation of the load balancing, a virtual environment was simulated with various avatars in it moving according to the random waypoint model with random wait. However, the choice of which waypoint go was not completely random. Three hotspots were defined, and there was a probability for the havatar choose one of these hotspots as its next waypoint. The purpose of this was to force an uneven distribution of avatars in the virtual environment, putting to test the algorithms of the proposed phase 2. Those who take account of the existence of hotspots form the regions on this basis, reducing the distribution overhead.

The environment consisted of a two-dimensional space, divided into 225 cells, forming a matrix of order 15. There were 750 avatars. All cells always belonged to some region, although they could be transferred between regions. There were eight servers $(S_1, S_2, ..., S_8)$, each one of them associated to a region, each of which could have 0 to all 225 cells. The capacity of all servers was different, with $P(S_i) = i \times 20000$. Thus, it was possible to test whether the distribution obeyed the criterion of proportional load balancing. Each session of simulated game was 20 minutes long. The total weight of the game was purposely set in a way that it was greater than the total capacity of the system, forcing the triggering of the load balancing.

In the graph of Figure 4, we can see that all the algorithms proposed met the proportionality criterion in load balancing. However, some of them introduced more overhead than the others. The reason for this is the fragmented regions. The lower the number of fragments that have the regions, the lower is this overhead because there are fewer boundaries between regions.

Observe that the algorithm ProGReGA is the one with the lowest overhead of all, as it was designed precisely to create the most possible contiguous regions, searching cells connected by the highest interactions, to minimize the algorithm overhead. The

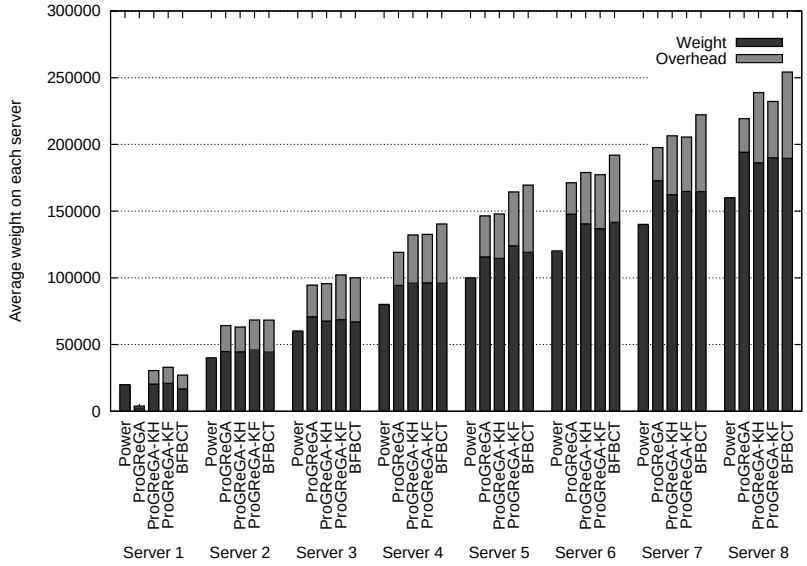

Fig. 4 Overall comparison of proposed load balancing algorithms

BFBCT algorithm, however, was developed in order to distribute the load based on a best-fit allocation, seeking to balance it as much as possible, ignoring, in phase 2, the existence of interactions between cells and regions. For this reason, the BFBCT was the one which created most fragmentation in the regions and thus most overhead between the servers.

Figure 5 shows the change in the total overhead on the server system, depending on playing time. It was observed that the overhead generated by each balancing algorithm varies relatively little with time, and the ProGReGA is the one which gives the lowest overhead in any moment of the game, and BFBCT has the largest overhead of all, for almost the entire simulation. The algorithms ProGReGA-KH and ProGReGA-KF alternate with intermediate values of overhead.

However, the introduction of overhead is not the only criterion considered. In figure 6 is shown how was the migration of users between servers throughout the simulated game session, for each algorithm used in phase 2. The value of migration has been divided in two: *walk migration*, which occurs when a player exchange server because he moved his avatar from one region to another, and *still migration*, which occur when a player exchanges servers without having moved. This kind of migration occurs because the cell where his avatar was has been transferred to another region as a result of a rebalancing of the load of the game. Walk migrations are more likely to happen when the regions are not contiguous.

We can see that ProGReGA is the one which has fewer walk migrations, precisely because their regions are contiguous. However, the fact of not trying to minimize the transfer of cells – the ProGReGA main objective is to minimize the overhead – the number of migrations is still the largest of all its variations, lower only than BFBCT, which was also the worst algorithm on the user migration criterion. The strategy of ProGReGA-KH and more strongly in the ProGReGA-KF to maintain the maximum

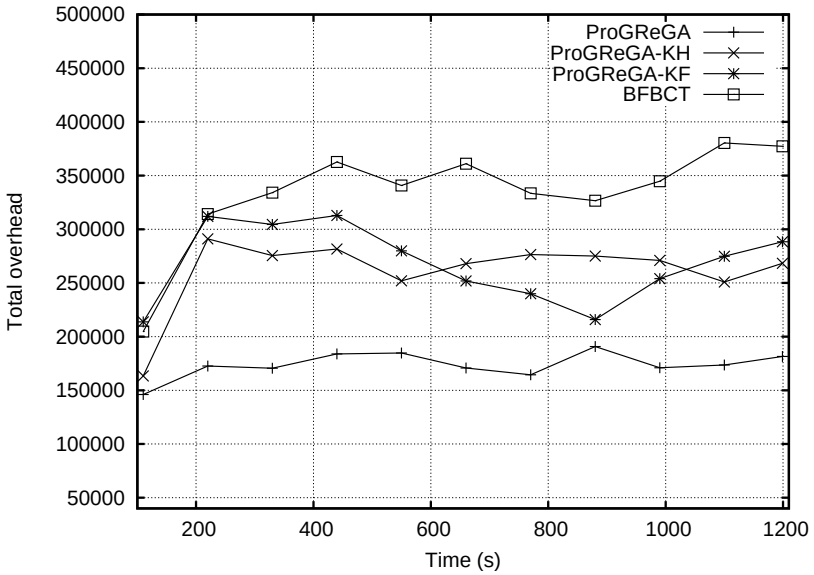

Fig. 5 Overhead introduced by each balancing algorithm during the game session

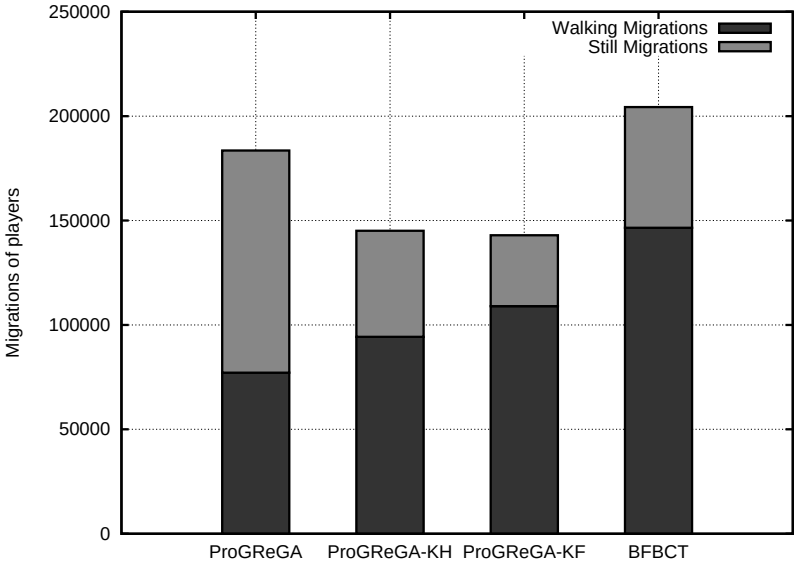

Fig. 6 Players migrating while still or walking depending on algorithm

possible cells during rebalancing, has as a result the lowest numbers of migration of players of all the simulated algorithms.

Another detail to be considered is the uniformity of distribution: all servers must have a usage rate as similar as possible so that the distribution is considered fair.

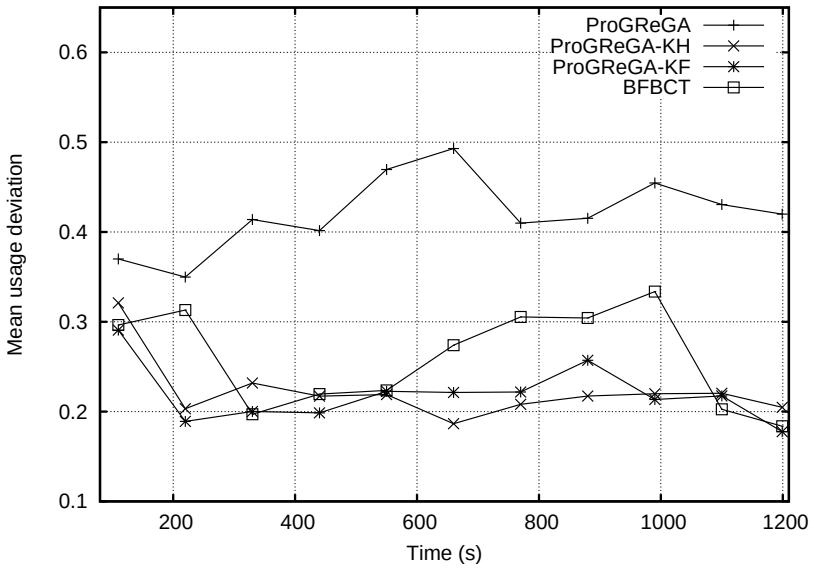

Fig. 7 Deviation of the ideal usage value

To measure this uniformity, the standard deviation, σ, was calculated for $u(S)$ of all servers for each simulated algorithm. Figure 7 shows the variation of σ in time.

Even with the servers providing the largest variation in the rate of use of its resources, the ProGReGA was the one with the least overall real weight (which includes overhead: $W_{total} + Overhead$) of all proposed algorithms. Its variant ProGReGA-KF had the least migrations of users, in exchange for an increase in overhead by 48%, on the average, compared to the original version of the algorithm. Which of these two is better depends on the specific game. In a real-time game , users contantly migrating between servers can introduce delay and hinder the interaction between players and the ProGReGA-KF, as shown, presented a total number of migrations 22% less than ProGReGA. Furthermore, it may not be possible to use some kind of "graceful degradation" to reduce the quality of the game, needing to save up the servers' resources. In this case, the ProGReGA would be the best option.

5 Conclusions and future work

It was proposed here a load balancing scheme for distributed MMOGs servers, taking into account the use of upload bandwidth of the server nodes. We considered important aspects such as quadratic growth in traffic when the avatars are close to one another, and the distribution overhead when players connected to different servers are interacting. The scheme, which is divided into three phases, proposed different algorithms for phase 2 (the balancing phase), and the ProGReGA presented the lowest overhead of all, while ProGReGA-KF presented the fewest migrations of players between servers.

As future work, the algorithms can be further refined, if the load balancing is considered for an average of the load in a period – in the last minutes, for example

– rather than the instantaneous value. Thus, it would avoid unnecessary rebalancing that would be caused by an oscilating weight.

Another possible future work is to create a load balancing scheme where there will be not only one server node, but a group of server nodes in each region. In this case, it can be investigated ways to balance the load of the system, considering this two-level distribution.

References

1. Schiele, G., Suselbeck, R., Wacker, A., Hahner, J., Becker, C., Weis, T.: Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming. Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid pp. 773–782 (2007)
2. Feng, W.: What's Next for Networked Games? Sixth annual Workshop on Network and Systems Support for Games (NetGames) (2007)
3. De Vleeschauwer, B., Van Den Bossche, B., Verdickt, T., De Turck, F., Dhoedt, B., Demeester, P.: Dynamic microcell assignment for massively multiplayer online gaming. Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games pp. 1–7 (2005)
4. Lee, K., Lee, D.: A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution. Proceedings of the ACM symposium on Virtual reality software and technology pp. 160–168 (2003)
5. Bezerra, C.E.B., Cecin, F.R., Geyer, C.F.R.: A3: a novel interest management algorithm for distributed simulations of MMOGs. Proceedings of the 12th IEEE International Symposium on Distributed Simulation and Real Time Applications, 2008, Vancouver, BC, Canada pp. 35–42 (2008)
6. Kernighan, B., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal **49**(2), 291–307 (1970)