Avatar Density Based Client Assignment

Lutz Behnke[1(✉)], Sven Allers[1(✉)], Qi Wang[2], Christos Grecos, and Kai von Luck[1]

[1] Department of Computer Science, University of Applied Sciences Hamburg, Hamburg, Germany
{lutz.behnke,sven.allers}@haw-hamburg.de
[2] School of Computing, University of the West of Scotland, Paysley, Great Britain

Abstract. Scaling the number of supported users for Massive Multi-User Games (MMOGs) allows more users to experience its content together. Supporting this capability needs consider the chain of all components that constitute the system between the client software of any two users.

A large body of research has been created over the last decades on the problem of dividing the resultant workload of a MMOG to specific nodes in a cluster of server. Connecting clients to the most appropriate server of this cluster is as important. Clients will place widely varying, dynamically changing requirements on processing, storage and network bandwidth resource on the MMOG.

We propose a novel mechanism of assigning clients to servers in a MMOG as part of an load balancing effort. It allows the optimization of resource utilization while being able to handle overload situations in the face of high avatar density and adapt to change over time.

Keywords: DVE · MMOG

1 Introduction

Managing the amount of processing, memory and network resources to provide the platform for any MMMOG follows the proven divide-and-conquer approach. Trying to scale up the number of players that can play the game concurrently a long list of bottlenecks have to be considered and overcome. The following work is part of a larger effort to create a complete MMORPG system that (a) will allow the creation of a single virtual environment with support for far more than 100k concurrent users; (b) will provide a non-stop service with no planed down-time, (c) will handle component failure (hard- and software) with minimal service degradation; (d) will allow unparalleled level of flexibility of the contained environment.

Prior publications have targeted the partitioning of the actual server workload, the resultant storage requirements and the aggregate network bandwidth that is required for the communication between the components (see [1]).

G. Wallner et al. (Eds.): ICEC 2016, LNCS 9926, pp. 1–12, 2016.
DOI: 10.1007/978-3-319-46100-7_12

The platform presented, QuP, will support a high number of concurrent players, as it allows large (>20) clusters of servers to maintain a single, continuous game environment.

QuP will also support a high local Avatar Density (the number of avatars in a given space), as it allows the same spatial area to be maintained on more than one server. This may lead to the next bottleneck: sending environment updates to the client computer, as this connection is usually the weakest link. To provide a minimal reduction user experience while greatly reducing the amount of change messages needed, we introduced Dynamic Budget Based State Aggregation [2]. While the concept is a generalized one, will it mostly benefit from situations that are present in may MMOs, where crowds of people or blocks of military units are moving in a similar fashion. Their action can easily be described as a group, with massive reduction of required messages.

QuP and DyBuBSA both benefit from an optimal assignment of clients to servers in order to group information about avatars that are in spatial proximity to a small subset of the whole cluster, idealy to a single server. This is due to the pre-fetching and pre-processing nature of both techniques. Client Assignment (CA) is a NP-complete problem [4], if encompassing a grouping problem it is even NP-hard [7].

Many existing solutions to CA in MMOGs are very application dependent. In many large-scale games, they range from the primitive, as in MMOBA-type (Massive Multiplayer Online Battle Arena) games like World-of-Tanks or StarCraft, where very small groups of players (usually 4–40) are grouped into a large number of world instances. These instances only exist for a short time (usually less than 60 min) while a single battle lasts. MOBA CA usually follows a few computationally simple rules to group players of similar strength in order to optimize the entertainment value of the game.

In MMO-RPGs, the assignment is often done via a static shard selection (e.g. World of Warcraft). This selection may be changed by the user, if the vendor allows it. The selection usually is subject to capacity planing by the service provider. These shards, or replicated instances of the virtual world can than be distributed statically to pre-planed server instances.

When attempting to construct a MMOG that models a continuous or at least connected space, in which all users interact in a single, integrated, continuous environment, the CA is more difficult to pre-plan and requires constant monitoring and update while the game is running. To increase the number of concurrently supported players within the world, MMORPGs often used zoning to partition the game population together within the game world. Unfortunately this often fails to support scenarios, where the whole population gathers explicitly to partake in a single event (e.g. massive battles in EVE online [3]).

We present a MMOG platform that is able to separate the spatial location of the avatar within the game environment from the Game Server (GS) to which it is assigned (e.g. QuP [1]).

Within it, large synergistic effects may be gained by grouping avatars in spatial proximity on the same server while avoiding overload situations before they develop. To minimize the required network traffic further, we introduce a method to reduce the average number of replicas of games objects on the GSs. We aim to achieve this goal by attempting to group avatars of similar Areas of Interest (AoI) on the smallest subset of servers as possible. To handle overload of a single server, QuP does allow for the same spatial area of the game to be represented by more than one server, albeit with a certain amount of additional network overhead.

Striking a balance between tight grouping and avoiding overload is the task of the Client Assignment Manager (CAM).

2 Related Work

A number of approaches to partitioning the client population prior to assigning them to servers have been discussed by commercial vendors as well as scientific publications. Most commercial vendors opt for primarily static or pre-computed assignment.

A wide range of work has been done on the subject of load distribution and load balancing. Most publications propose the partitioning of the game world and thus the avatar population. These partitions are then assigned to servers within the system. [13] proposes to assign one region to one server, with the region being partitioned further into cells, which can be ceded to other servers in high load situations. The load information of each server are regularly submitted to a central load collector server. As [8] observes, the cells are only moved to neighboring servers and global state is not considered, thus underutilized servers may be ignored since they are not adjacent to the server in overload. To order to address these shortcomings, [8] proposes a hybrid approach in which the load is shed not only to neighbors, but appropriate servers are also chosen from a global set.

The work in [10] approaches load balancing via linear optimization, looking not only at server processing load, but the required communication overhead among the servers as well. The aim is to distribute the load evenly among servers while minimizing communication at the same time. They construct a graph based on the Area of Interest (AoI) of each avatar, connecting the graph when the AoIs overlap. [11] improve this by using an ant colony optimization algorithm, but also showed that there is a minimal size of avatar population and server cluster that limites the cost efficiency of more complex load balancing approaches.

[4] does not attempt to distribute load evenly, but rather to ensure that pre-defined levels of Quality of Service (QoS) are met by each server. These QoS include CPU utilization, network bandwidth consumption. They propose an appropriate partitioning and merging scheme.

Static partitioning is the state of the art of many commercial online games, but does not match the dynamics of an MMO-RPG. [5] investigate a number of algorithms and their fitness for approaching optimal distribution of clients to server. They show linear programming to be able to provide an optimal solution, but also that the required cost of computation is prohibitive for practical use.

[15] show that all the other algorithms are costly as well. In [6] they propose a method that uses a simulated heat flow model, based solely on local information.

All above approaches assume that all client connections and thus avatars produce a similar load and thus partition the avatar population by partitioning the environment space. A different approach is taken by [12]. Here a server is focused on the avatars that are assigned to it. The scheme first sorts the avatars by the number of AoIs they are present in. It then repeatedly moves the most interesting avatar, until a number of iterations is reached.

[9] states that regions do not reduce the inter-server communication to a sufficient degree. Instead they favor a simple load balancing scheme. They propose the use of a NAT-Server between server and client and the use of standard load balancing methods like round-robin. The proxy approach is also proposed by [14] to isolate the region-to-server assignment from the client-to-server assignment.

3 Requirements

As this work is based on QuP platform for large scale MMOGs, the AoI of each avatar is represented by its view box, an Axis Aligned Bounding Box (AABB). The CAM should group the assignments to the game servers according to the spatial grouping of their avatars within the virtual environment. This is expected to allow a reduction of retrieved state and subsequent state update messages to a given server as multiple avatars will access the same pre-fetched object and environment data through a local lookup rather than costly remote query.

The CAM must also ensure that the CPU load on a given server is within acceptable parameters. As the load created by a single client connection can vary widely, this will require feedback from the GS. Available main memory and network bandwidth must be considered as well.

Based on the requirements, we derive the following cost function to govern the client assignment. The client will be assigned to the server that will return the minimal value from the cost function below:

$$cost(a, s) = w_0 * -g(a, s) + w_1 * Load(s) - w_2 * MemFree(s) \tag{1}$$

For an avatar a and the serve s the function $g(a, s)$ will calculate the Grouping Factor (GF). The GF represents similarity of the AoI of the new avatar to the AoI of the avatars that are already present on a given server. It is assumed that most of the objects in the AoI of the other avatars have already been loaded and that further updates on those objects will be relevant to both the existing and the new avatars. The actual computation of the GF will be discussed in Sect. 4.

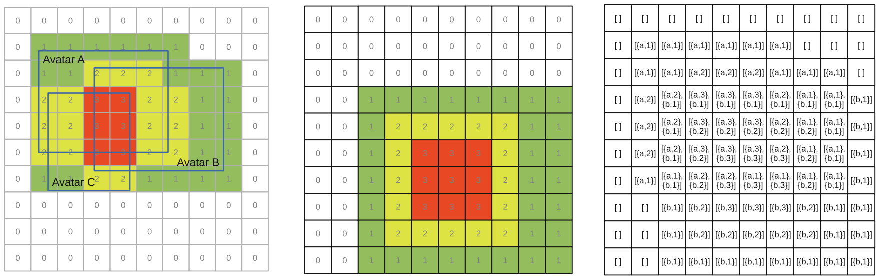
Memory requirement is not given as the percentage used, but the amount of remaining memory instead, so that servers having large amounts of memory remaining are favored, regardless of their total amount of memory. w_0, w_1, w_2 are weights to control the priority of the potentially contradictory requirements of maximizing the grouping factor and not surpassing limits on CPU load and memory utilization. Network utilization is not considered as separate parameter here, as it directly relates of the grouping factor. More fine grained modeling on this may be an area for further study.

As the avatars move over time, the client assignment must be able to track validity of the calculation results over time. This may lead to the need for a reassignment of clients to new GS over time. Also any CAM solution should be able to scale with the MMOG to support a user population far in excess of 100k concurrent users, including the resultant churn, expected clustering of user log-in and number resultant number of GS.

4 Proposed Solution

We propose calculating the grouping factor, using a heat map. To construct the heat map, the virtual environment is partitioned into a set of blocks of equal size. This is similar to partitioning of the actual game environment into micro cells [5], but we use these blocks only for the grouping calculation. Due to the architecture of QuP, this is done in three dimensions. The diagrams below are restricted to two dimensions for the benefit of illustration only.

As the AoI of an avatar governs which objects in the game have to be pre-fetched to a server, we use the view boxes and their overlap to determine the grouping factor when considering a server for assignment.



(a) Viewbox overlap and resultant heatmap for server A

(b) Heatmap for server B

(c) Combined heatmap for servers A and B

Fig. 1. View box overlap

Each block holds a set of servers that serve at least one avatar whose view box overlaps with the block. A tally of the number of intersecting view boxes is kept, as shown in Fig. 1c

In QuP and the LB all object coordinates are within a norm cube ($0 <= x, y, z < 1$). This cube is partitioned into n blocks per axis. For each block (identified by it coordinate between 0 and n-1) and each server, $T(x, y, z, s)$, a tally of avatar view boxes that overlap this block is maintained. This allows the calculation of the grouping factor for each server in a block.

As each client is requesting a server assignment, the view box of the avatar is queried from the world state (maintained by the QuP servers). As the server is chosen, its tally is updated in each of the blocks that the clients view box intersects with. As a server is considered to cover the view box of an avatar even if the view box only intersects the spatial area covered by the server, this method forms an overestimating heuristic. It allows the computation of the GF used in Eq. 1.

4.1 CAM Components

The CAM is partitioned into two different components: the Load Balancer (LB) and the Server Monitor (SM). In our experiment we only used a single LB instance (for further scalability see Sect. 7). One SM is assigned to each GS. See Fig. 2 for communication links and integration into the full system. The actual CAM components are shown in darker color.

As the central component, the LB holds the actual heat map and meta information on the known servers and the previously assigned clients. The Authentication Server queries the LB to receive a server assignment. The LB in turn maintain a lease on the client-server assignment to update the tallies for the heat map. This lease will be confirmed if an acknowledgement messages is received by the LB. This message will sent by the server once the client has connected to it.

The LB continuously monitors the servers for any messages send to it. This may either be heart beat messages by the server or responses to a client assignment requests. Should a server not have sent any message longer than a configurable threshold, it is considered dead and will be removed completely from the state of the LB. It may be reintroduced only by administrator command. If there are pending assignments that have not been confirmed yet, they are rolled back in the heat map as well.

The SM running in each GS will inform the LB of any successful client connections. As the avatars move through the environment, the SM will track which heat map regions are being covered and uncovered by the shifting view boxes of each avatar. The list of changes is reduced by applying inverse operations (e.g. the number of new avatars in a region is subtracted from the number of avatars that have left the region). The resulting difference list is sent to the LB in regular intervals.

4.2 Base Grouping Factor

The assignment algorithm has to weigh between placing an avatar on a server that already covers a spatial region or placing it on a new server to avoid overloading the former server. There are two special cases to consider when selecting servers for the client: (a) empty servers (i.e. without any clients) and (b) server whose covered area does not intersect with the avatar view box. Empty servers will not only occur after starting the system, but can also result from avatars congregating on small spatial areas of the game, leaving others completely empty and thus the corresponding servers idle, as reassignment incrementally groups the avatars on fewer and fewer servers to optimize the grouping factor.

Both cases are similar, in that they result in a grouping factor of 0, as no view boxes overlap. The client assignment would only be controlled by the CPU and memory resource utilization values.

We propose to use a grouping factor for empty servers, that represents the state in an idealized game, containing only equidistant avatars that are evenly distributed among the servers. For this we calculate a Distance value (for a world in 3 dimensions) from an ideal field count to an average field count of the actual servers in the running game:

$$IdealFieldCount = g_0 \frac{P^3}{ServerCount} + g_1 \frac{UsedFields}{ServerCount}$$

The heat map is split into precision P, given as the number of regions per dimension g_0, g_1 are weights so that $(0 < g \leq 1, g_0 + g_1 = 1)$. The weights represent the balance between two objectives: the even partitioning of the whole virtual environment among the servers and the even distribution of the regions already used among the servers. The optimal values are application specific.

From the ideal field count, a distance value can be computed.

$$Distance = \frac{\sqrt[3]{IdealFieldCount} - \sqrt[3]{AvgFieldCount}}{2} * \frac{1}{P}$$

As a large distance would be bad for grouping avatars, the grouping factor for empty servers is $g(a, s) = -Distance$

All server that are neither empty nor have a server view box that overlaps with the avatar view box are considered in turn for assignment by setting the cost to the Manhattan Distance from the fields that contain the avatars view box. The position of a given non empty server is calculated for each field it is assigned in.

4.3 Reassignment

After each avatar is assigned to a server, it is likely to move through the virtual environment. Thus the grouping factor of the servers will deteriorate over time. To counter this, our approach supports a protocol for client reassignment. Executing this protocol produces a small overhead. Reassignment may result on a negativ impact on the user experience, even if only for a limited time. Thus a certain amount of hysteresis is required to regulate the frequency of reassignments of a given client connection.

In order to track the quality of each assignment, as well as to monitor a user session, the LB maintains a (lightweight Erlang) monitor process for each connected client. In regular intervals, each process is activated in order to recalculate the server assigned to the client. Should the calculation result in a different server, the client is reassigned.

As the new avatar will shift the aggregate view box of the newly assigned server, the probability of assigning the same server on the recalculation increases. If needed, an additional weight against reassignment could easily be applied to the calculation.

The server will notify the LB when the client has disconnected. This will remove the monitoring process for the client session.

If the distance is greater than a pre-set threshold, a client reassignment process will be triggered. If the assignment results in a different server, the re-connection protocol is executed (see Fig. 3b).

The LB will also re-check the assignment of each avatar after a pre-set interval. This will slowly, but continuously, improve the assignment in the face of avatar movement. While this will regularly query the position of the avatar from the QuP subsystem, it will not suffer the very large bandwidth requirement of tracking each position change of each avatar.

A further method of initiating an assignment reevaluation will be triggered by the server directly. If it detects insufficient CPU, memory or bandwidth resources, it will send an emergency signal to the LB to trigger immediate assignment re-evaluation.

The reassignment protocol will notifiy the new server to start pre-fetching all object data required by the avatar. Then the client is informed by the old server of the need to switch. The message will include the contact details of the new server as well as a token string that allows the client to authenticate to the new server.

5 Experiments

We have implemented a prototype of each of the component's shown in Fig. 2. For the tests, the AS and the game clients were replaced by a test server in order to isolate the LB for measurements. Also, the LB was configured to contact a special QuP mock server in order to separate the time required for computing the server assignment from the time for the QuP look up. The mock was pre loaded with the position of the avatars, provided no redundancy and was co-located with the LB on the same maschine.

The test were done on three PC type virtual machines. The VM manager was configured to allow full CPU utilization during the test. The systems where running Erlang version OTP 18 on Ubuntu Linux.

Tests reflect the three types of avatar distribution introduced by [10]. We consider them sufficient representations of avatar behaviour in an MMOG. The distributions were adapted to three dimensions as supported by QuP. All avatars are place in a norm cube. Skewed distribution was set to 60 % of avatars in a cube occupying the lower 0.3 of each dimension; clustered set to 75 % of the avtars being in 10 evenly distributed groups; the remaining avatars were uniformly distributed in the whole environment.

Two test series with 10,000 and 100,000 avatars were set up. The heatmap was partitioned with a precision of $\mathrm{P} = 10$ blocks per dimension. The smaller scale

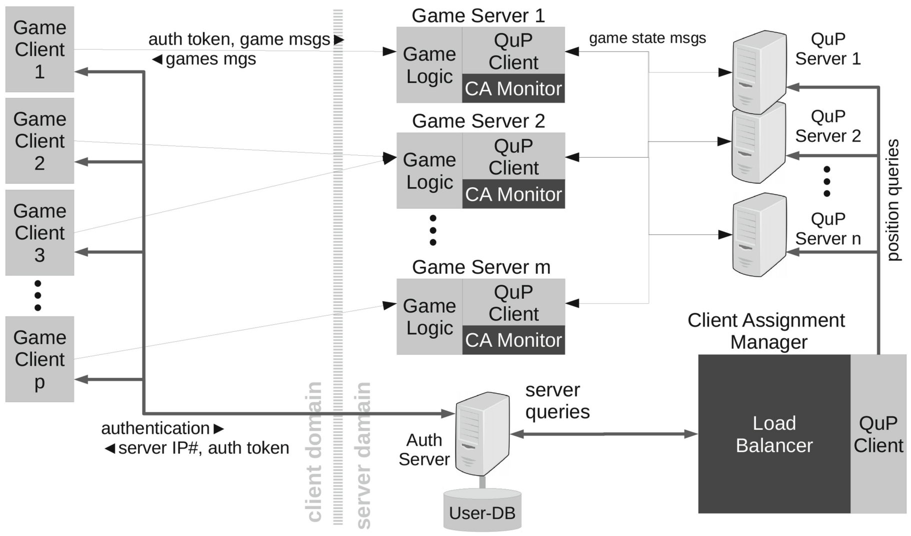


Fig. 2. proposed timadorus architecture

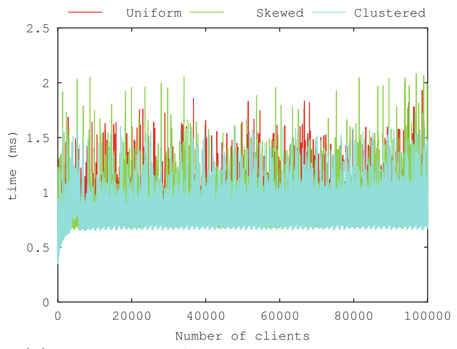
test was conducted with only three servers in order to allow manual verification of the assignment. The larger one to test the time requirements based on the number of clients previously assigned. The larger scale test used seven servers to inspect the increase of computation time for an increased number of avatars.

Increasing P results will an cubic increase the memory requirements of the LB. We assume that P = 10 will partition the space sufficiently to effectively distribute clients to a number of game servers in the order of 10 to 100. Any installation that outgrows this, would should have enough resources to support much higher values of P.

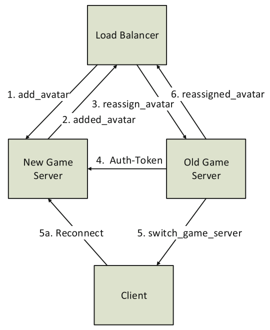
6 Results

Figure 3a, shows the 99.9th percentile for the time elapsed from a placement request being sent to reply given by the LB. It uses P = 10, thus 1000 blocks. The measurement does not include the network communication to the QuP servers. We observe a constant time requirement once the data structures have been filled.

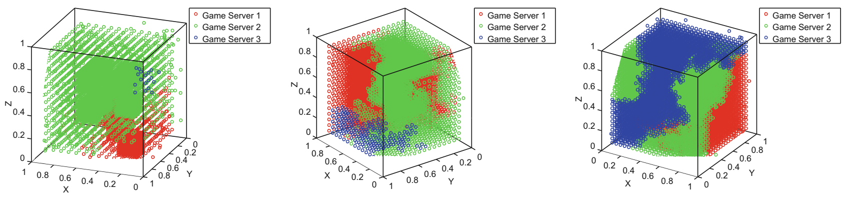
Figure 3 shows the distribution of the avatars to three servers in the smaller scale test environment for each of the three distributions.



(a) Reply times for all distributions, P=10



(b) Reassignment process



clustered skewed uniform

Fig. 3. Avatar grouping (P = 10, 3 servers, 1000 avatars)

7 Conclusion and Future Considerations

We have shown that, in combination with QuP [1] and DyBuBSA [2], our approach to client assignment provides the needed capabilities to drastically scale up the number of supported concurrent users in an MMOG with a single continous game environment, or distributed virtual environment in general.

For a given set of servers, the load balancer will return an assignment in constant time, regardless of the number of avatars currently active. The only prerequisite is a sufficient amount of memory to maintain the heatmap, but this is constant and can be controlled by the number of blocks per dimension.

The quality of the assignment will improve the utilization rate of the prefetched data on the target server in the normal case. The worst case, if data in the LB has drifted too much, an less than optimal assignment is given, further increasing the drift distance. If a sufficient drift distance has been detected, a reassignment will be initiated for all avatars concerned.

Sub-optimal assignments will only cause the CPU, memory and bandwidth resources of the server to be depleted at a lower number of client connection compared to the optimal case. For a given number of expected supported client connections, this will either require more GS nodes to be started, or the maximum number of supported clients will be redcued. For the clients already connected, even in the face of sub-optimal assignment full quality service will be proved.

The reassignment process will minimize the impact on the user experience, but may create small interruptions as the network connection is switched from one server to another.

When scaling up the system, the number assignment requests and reassignment protocol messages will increase with the number of connected game servers and the number of clients connecting to the system concurrently. This will result in an increased network bandwidth requirement placed on the LB. Access to the data structure that holds the heatmap and its ancillary data may pose a bottleneck, since it is held on a single compute node and not replicated. It receives updates sent by the monitor processes of each server, as well as all the client assignment requests. More importantly, creates a single point of failure for the whole system with regards to connecting new users and the continuous improvement of the resource utilization for the already connected clients.

We intend to undertake further work on eliminating this source of system failure and to support even larger user bases. One possible method is to use a Distributed Hash Table (DHT) to maintain the heat map in a small set of servers, as the changes to the map are generally small and localized. The authentication server would contact the LBs in a round robin fashion. Thus the ASs can itself be duplicated and contacted by the clients in a round robin fashion themselves. The re-calculation processes and possible re-assignment processes can be separated onto further server nodes as well.

References

1. Behnke, L., Grecos, C., Luck, K.V.: QuP: graceful degradation in state propagation for DVEs. In: Proceedings of International Workshop on Massively Multiuser Virtual Environments. ACM Press, Singapore (2014). http://dl.acm.org/citation.cfm?id=2577389
2. Behnke, L., Wang, Q., Grecos, C., von Luck, K.: Budget based dynamic state update aggregation. In: Proceedings of the 7th ACM International Workshop on Massively Multiuser Virtual Environments - MMVE 2015, pp. 25–26, Portland, Oregon (2015). http://dl.acm.org/citation.cfm?doid=2723695.2723696
3. CCP Manifest: A Weekend of Epic Destruction in EVE Online (2013). http://community.eveonline.com/news/dev-blogs/a-weekend-of-epic-destruction-in-eve-online/
4. Chen, J., Wu, B., Delap, M., Knutsson, B., Lu, H., Amza, C.: Locality aware dynamic load management for massively multiplayer games. In: Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming - PPoPP 2005, p. 289, New York, USA (2005). http://dx.doi.org/10.1145/1065944.1065982, http://portal.acm.org/citation.cfm?doid=1065944.1065982
5. De Vleeschauwer, B., Van Den Bossche, B., Verdickt, T., De Turck, F., Dhoedt, B., Demeester, P.: Dynamic microcell assignment for massively multiplayer online gaming. In: Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games - NetGames 2005, p. 1, New York, USA (2005). http://portal.acm.org/citation.cfm?doid=1103599.1103611
6. Deng, Y., Lau, R.W.H.: Dynamic load balancing in distributed virtual environments using heat diffusion. ACM Trans. Multimedia Comput. Commun. Appl. **10**(2), 1–19 (2014). http://dl.acm.org/citation.cfm?doid=2579228.2499906

7. Falkenauer, E.: A new representation and operators for genetic algorithms applied to grouping problems. Evol. Comput. **2**(2), 123–144 (1994). http://www.mitpressjournals.org/doi/abs/10.1162/evco.1994.2.2.123
8. Lau, R.W.: Hybrid load balancing for online games. In: Proceedings of the International Conference on Multimedia - MM 2010, p. 1231, New York, USA (2010). http://dl.acm.org/citation.cfm?doid=1873951.1874194
9. Lu, F., Parkin, S., Morgan, G.: Load balancing for massively multiplayer online games. In: Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games - NetGames 2006, p. 1, New York, USA (2006). http://doi.acm.org/10.1145/1230040.1230064, http://portal.acm.org/citation.cfm?doid=1230040.1230064
10. Lui, J., Chan, M.: An efficient partitioning algorithm for distributed virtual environment systems. IEEE Trans. Parallel Distrib. Syst. **13**(3), 193–211 (2002)
11. Morillo, P., Fernandez, M., Orduna, J.: An ACS-based partitioning method for distributed virtual environment systems. In: Proceedings International Parallel and Distributed Processing Symposium, p. 8, No. C, IEEE Comput. Soc. (2003). http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1213283
12. Morillo, P., Orduña, J.M., Fernández, M., Duato, J.: An adaptive load balancing technique for distributed virtual environment systems. In: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PCDS 2003), vol. 15, pp. 256–261 (2003). http://www.scopus.com/inward/record.url?eid=2-s2.0-1542537865&partnerID=tZOtx3y1
13. Ng, B., Si, A., Lau, R.W., Li, F.W.: A multi-server architecture for distributed virtual walkthrough. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology - VRST 2002, p. 163. New York, USA (2002). http://portal.acm.org/citation.cfm?doid=585740.585768
14. Quax, P., Cornelissen, B., Dierckx, J., Vansichem, G., Lamotte, W.: ALVIC-NG: state management and immersive communication for massively multiplayer online games and communities. Multimedia Tools Appl. **45**(1–3), 109–131 (2009). http://www.springerlink.com/index/10.1007/s11042-009-0299-3
15. Deng, Y., Lau, R.W.H.: On delay adjustment for dynamic load balancing in distributed virtual environments. IEEE Trans. Vis. Comput. Graph. **18**(4), 529–537 (2012). http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6165133