

Multi-Server Dynamic Load Balancing for Networked Virtual Environments

Jehn-Ruey Jiang, Fu-Hsiang Chang
Department of Computer Science and Information Engineering
National Central University, Jhongli City, Taiwan
jrjiang@csie.ncu.edu.tw, majuschang@gmail.com

Abstract

This paper proposes a cell-based dynamic load balancing (LB) scheme, namely Directed Load Diffusion (DLD), to achieve load balance among servers in a networked virtual environment (NVE). The virtual environment is divided into small hexagonal areas called cells, and a server is responsible for managing a region consisting of many adjacent cells. Different servers have different capacities, and the load degree of a server is the utilization of its capacity. The DLD scheme is to keep server load degrees as even as possible without compromising performance. We perform extensive simulation experiments for the DLD scheme and compared it with two relevant cell-based LB schemes, namely Ahmed and ProGReGA, to show its advantages.

Keywords: Load Balancing, Networked Virtual Environment, Hotspot, Massively Multi-player Online Game.

1. INTRODUCTION

In a *networked virtual environment (NVE)*, players or users can assume roles of *avatars* to navigate in a computer generated realistic virtual world or *virtual environment (VE)* to interact with one another through networked links. NVEs are widely applied to a variety of areas like military simulation, education, training and network games. The *massively multi-player online game (MMOG)*, such as World of Warcraft (WoW) [7], which is nowadays a billion-dollar business, is a typical example of NVEs. Thousands or tens of thousands of NVE users may log on and navigate throughout the VE concurrently. Numerous servers are provided to cooperatively manage the users by receiving action messages from them, simulating the game, and sending game state update messages to relevant users. However, the dynamic actions of users may lead to load unbalance among servers. Several *load balancing (LB)* schemes were thus proposed to flexibly balance NVE server loads [1-6].

LB schemes can be classified into four classes: shard-based, zone-based, partition-based, and cell-based. The shard-based LB scheme, as adopted by WoW [7], makes multiple VE duplicates (called shards or realms) run concurrently, with each duplicate being supported by a separate unified sever cluster. The overloaded situation of servers can be avoided by explicitly limiting the maximum number of users in a shard. When the number of users grows and exceeds a pre-specified threshold, a new duplicate is produced and supported by a new server cluster. The shard-based LB schemes have low computation overheads; however, they have one limitation that avatars in different shards cannot interact with one another.

The zone-based LB scheme divides the whole VE into fixed-sized zones or regions, each representing a country or city and managed by a server. Users have the chance to interact with one another

since there is only a single shard. However, the passage from one zone to another requires special portals (e.g., special tunnels) or teleporting. Since users tend to gather around hotspots where certain events occur and/or NPCs (non-player characters) appear, the server managing many hotspots may easily become overloaded.

The partition-based LB scheme, such as the kd-tree scheme and the schemes adopted by QuON [5] and VSO [6], divides the whole VE into various-sized partitions, each managed by a server. The VE partitioning is dynamical. For example, the kd-tree scheme, QuON and VSO uses the kd-tree, Quad-tree, and Voronoi-diagram structures, respectively, to grow or shrink partitions. Partition-based LB schemes have the advantage that the load of servers can be adjusted dynamically and flexibly to balance the server loads evenly; they have the disadvantage of high computation overheads, though.

The cell-based LB scheme, such as Ahmed [2] and ProGReGA [3], divides the whole VE into small *cells* (or *microcells* mentioned in some papers), a group of which is managed by a server. Typical cell shapes include triangles, squares, and hexagons. Different heuristic strategies are developed for assigning servers to manage different groups of cells. The cells managed by a server may or may not be contiguous and may be of any numbers; therefore, the load of an overloaded server can dynamically and flexibly be transferred to arbitrary other servers. The computation overheads of cell-based LB schemes are mediate.

This paper proposes a cell-based LB scheme, called *DLD (Directed Load Diffusion)*, which partitions the whole VE into hexagonal cells to assign a region of adjacent cells to be managed by a server. Different servers have different *capacities*, and the *load degree* of a server is the utilization of its capacity. The DLD scheme is to keep server load degrees as even as possible, while maintaining low load transfer overheads, low inter-server communications, and low load deviation ratios. We perform extensive simulation experiments for the DLD scheme and compare it with two relevant cell-based LB schemes, namely Ahmed [2] and ProGReGA [3], to show its advantages.

The rest of this paper is organized as follows. In Section 2, we introduce some related work. Section 3 elaborates the proposed scheme, and Section 4 presents its simulation results and comparisons. Finally, some concluding remarks are drawn in Section 5.

2. RELATED WORK

In this section, we introduce two cell-based LB schemes, namely Ahmed [2] and ProGReGA [3], which are most related to the proposed scheme. Below, we describe the schemes one by one.

The Ahmed scheme [2] measures the load of a server by the sum of message rates of all players the server must handle. A server whose sum of message rates exceeds a pre-specified threshold is regarded as overloaded. To balance the load, the Ahmed scheme

finds all clusters of adjacent cells managed by the overloaded server. The smallest cluster is selected and, from this cluster, the cell which has the least interaction with other cells of the same server is first selected to be transferred to the least loaded server. The selection process continues until the server is no longer overloaded or there is no server that can take over the transferred loads.

The ProGReGA (proportional greedy region growing algorithm) scheme [3] assumes that the whole VE is divided into regions, each of which consists of several cells and is managed by a server. Given a list of the regions to be rebalanced, all servers managing these regions share proportionally the loads of all regions according to the server power. Initially, the heaviest-loaded cell is selected to be the first cell of a region to be managed by the most powerful server. Afterwards, a neighbor cell adjacent to the heaviest-loaded cell which has the largest inter-cell interaction overheads is added into the region. The cell-addition continues until the most powerful server has shared the proportional loads. Then, the process continues to form the second region to be managed by the second powerful server by repeatedly executing the cell-addition process. Iteratively, the third region, the fourth region, ..., are formed until every cell is added into one region to be managed by one server. One undesirable effect of the ProGReGA scheme is that after rebalancing, one or more regions may completely change the servers originally managing them, causing high load transfer overheads.

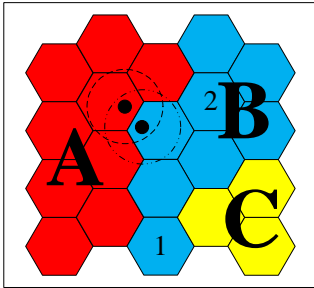


Figure 1: The hexagonal partitioning of a virtual environment.

3. PROPOSED SCHEME

3.1 Assumptions and Definitions

The whole SE is assumed to be divided into hexagonal cells, and a region containing some cells, not necessary adjacent, is managed by a server. An avatar (i.e., player) navigating in the VE performs some actions and sends associated *action messages* to the region sever. The server then updates the VE state according to the actions and sends *update messages* to every avatar of which AOI (Area of Interest) includes the updates, where an avatar's AOI stands for the circular area of a fixed radius or range centered at the avatar. For example, black dots in Figure 1 represent avatars and dashed circles represent their AOIs. Two regions are adjacent if they have adjacent cells; two servers are adjacent if they manage adjacent regions. For example, regions A and B in Figure 1 are adjacent, but regions A and C are not adjacent. Two servers need to exchange messages if one server manages some boundary cells with avatars whose actions cause the updates included in the AOI of avatars in the cells managed by the other server. For example, in Figure 1, the server managing region A and the server managing region B need to exchange messages since they have avatars that are managed by different servers and influence each other.

It is assumed that the overall load of a sever comes from three aspects: (a) the *Action Load (AL)*, the load to process action messages sent by avatars, (b) the *Computation Load (CL)*, the load to compute the updates caused by action messages, and (c) the *Update Load (UL)*, the load to send update messages to relevant avatars. A server has a pre-specified *Capacity (CAP)* to take loads. In this paper, we define the *Server Load (SL)* to be the load of a server and the *Load Degree (LD)* to be the utilization of the server capacity, i.e., the ratio of the server load to the server capacity. Specifically, $LD = SL/CAP$.

To judge the load condition of a server, we have three thresholds for the load degree, as explained one by one in the following.

- (a) *Overload Threshold (OLT)*: A server is *overloaded* if its load degree exceeds OLT. An overloaded server can be a *giver server* to transfer loads to other servers.
- (b) *Safety-Load Threshold (SLT)*: A server is *normally loaded* if its load degree is between SLT and OLT. By SLT, we can derive two useful server attributes, the *Safety Capacity (SC)* and the *Giving Load (GL)*, according to the following calculations: $SC=(SLT-LD)\times CAP$ and $GL=(LD-SLT)\times CAP$. Note that when LD is larger than SLT, SC is defined to be 0. The SC of a server indicates the maximum extras loads that can be transferred from other servers into the server.
- (c) *Underload Threshold (ULT)*: A server is *lightly loaded* if its load degree is between ULT and SLT; it is *underloaded* if its load degree is beneath ULT. A normally loaded or a lightly loaded server can be a *taker server* to take over loads from other servers. The ULT threshold is very useful for performing LB at the presence of hotspots, which will be described later.

Note that the server load degree and the above-mentioned threshold values are between 0 (no load) and 1 (full load) and $OLT > SLT > ULT$. For a specific server, we below define the *Local Load Degree (LLD)*, to estimate the server load conditions for its neighborhood of itself, indexed by 0, and its adjacent n neighbor servers, indexed by $1, \dots, n$.

$$LLD = \frac{\sum_{i=0}^n SL_i}{\sum_{i=0}^n CAP_i}$$

3.2 The Proposed Algorithm

In this subsection, we describe the proposed *Directed Load Diffusion (DLD)* scheme, which is a greedy load balancing algorithm trying to keep every server *normally loaded*. A server can be a *taker server* to take over other servers' loads if it is lightly loaded or underloaded (i.e., its load degree is beneath SLT). A server X checks its load degree every *Check Load Period (CLP)*. If X is overloaded, it then performs *General Load Balancing (GLB)*, described below. X first calculates LLD to evaluate the load conditions for its neighborhood of itself and all its neighbor servers. If the calculated LLD is smaller than SLT, then X 's neighborhood is assumed to be lightly loaded (or underloaded) and X can transfer its load to the neighbor server Y with the lowest server load (i.e., SL_Y). The load transferred from X to Y is $Min(GL_X, SC_Y)$. Otherwise, X 's neighborhood is assumed to be heavily loaded. It is probably that every neighbor server of X has a load degree exceeding or approximating SLT. In that case, X transfers its load to the neighbor server Z with the largest safety

capacity (i.e., SC_Z). The load transferred from X to Z is $\text{Min}(GL_X, SC_Z)$. Note that X transfers its load to only one server for GLB within a CLP; if X is still overloaded after the load transfer, X will transfer its load within next CLP.

It is likely that all servers in the neighborhood of a hotspot have load degrees exceeding SLT so that an overloaded server X may not transfer its load to any other server for GLB. In that case, X performs *Forced Load Balancing (FLB)* by sending an FLB Request (FLB-Req) message to all neighbor servers. When server Y receives the FLB-Req message sent by X , Y is forced to transfer its load to its neighbors so that Y is lightly loaded to take over X 's load. To prevent Y from transferring its load to a server that is X 's and Y 's common neighbor, the FLB-Req message sent by X contains a *Forbidden Transfer List (FT-List)* including X and all neighbors of X . Y transfers its load to its every neighbor server not in FT-List according to the “*high safety capacity first*” order. The load transferred from Y to Z caused by X 's FLB-Req message is $\text{Min}((LD_Y - ULT) \times CAP_Y, SC_Z)$ so that Y becomes lightly-loaded and Z becomes normally-loaded after the load transfer. If Y cannot make itself lightly-loaded, Y forwards X 's FLB-Req message to neighbors not in the FT-List to further make them underloaded. However, if all Y 's neighbors are in the FT-List, Y just stops FLB. Note that Y 's neighbors are added into the FT-List before Y forwards X 's FLB-Req message.

We have above described when and how much to transfer load between servers. Below, we describe how to select cells for load transfer. In practice, a server transfers its load by reducing the number of cells managed by it. For better performance, the server tries to keep all the cells adjacent but not to be of the shape of narrow strips, which may cause more inter-server communications. We use the *Cell Weight (CW)* to help cell selection. The CW of a cell c is defined to be the ratio of the number of c 's adjacent cells managed by different servers to the total number of c 's adjacent cells. To take Figure 1 for example, cell 1 managed by server B has the CW of 2/3, while cell 2, 1/6. The larger the CW of a cell is, the higher the priority of the cell to be selected as a cell candidate (CC) to be transferred to another server.

There are two options of the DLD scheme: DLD with Load Constrains (DLD-wLC) and DLD without Load Constrains (DLD-woLC). DLD-wLC has the limitation that the load involved in a CC should not exceed the load to be transferred in GLB or FLB, while DLD-woLC does not have the limitation. The former (resp., the latter) can (resp., cannot) prevent the load of the taker server from exceeding SLT too much but makes LB more inflexible (resp., flexible).

4. SIMULATION

4.1 Simulation Settings

We limit a server to manage at least one cell but not limit the maximum number of cells that a server can manage. We assume there are 8 servers and 800 avatars. Furthermore, there may be 3 hotspots or no hotspot in the VE. OLT, SLT, and ULT are set as 0.9, 0.8, and 0.6, respectively. We set the server capacity to be 1600 units and the load caused by avatar actions to be arbitrarily 3, 5, or 10 units. The AOI radius and the cell side length are both set to 4 units. The avatar speed is assumed to be 0 to 2 units per second, and the avatar follows the random waypoint mobility model with 50% of choosing the direction toward the closest hotspot. The CLP is set as 32 seconds. Table 1 shows all the simulation settings.

Table 1: Simulation Settings

Parameter	Value
The number of cells	224 (14×16)
The number of servers	8
The number of avatars	800
Server capacity	1600
Action load	1
Update load	2
Computation load	3, 5, or 10 (random)
AOI radius	4 units
Cell side length	4 units
Avatar Speed	0~2 units (random)
Check load period	32 sec
Simulation duration	30 min

4.2 Comparisons

We simulate the proposed DLD scheme and compare its two options (i.e., DLD-wLC and DLD-woLC) with two related schemes, namely the Ahmed [2] and ProGR-eGA [3]. The comparisons are shown below in terms of the following overheads: avatar migration, inter-server communication, and load deviation ratio, each in a separate subsection.

4.2.1 Avatar Migration

Avatar migration has two portions: walking migration and still migration. When an avatar navigates in the VE and walks from one cell managed by a server to another cell managed by another region, the *walking migration* occurs and the avatar's data should be transferred from the former server to the latter. When servers perform LB to transfer loads of some cells from one server to another server, the *still migration* occurs and all data of the avatars in the cells should be transferred from the former server to the latter. By Figure 2, we can observe that both DLD-wLC and DLD-woLC outperforms the other two schemes in terms of walking migration, still migration, and total avatar migration whether there are hotspots or not.

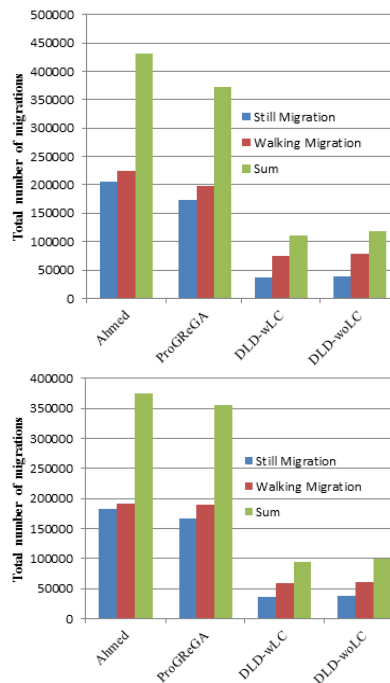


Figure 2: Avatar Migration Comparisons of LB Schemes with hotspots (up) or without hotspots (down).

4.2.2 Inter-server Communication

As we have mentioned, two servers need to exchange messages if one server manages some boundary cells with avatars whose actions cause the updates included in the AOI of avatars in the cells managed by the other server. Such a case causes the overheads of inter-server communication. By Figure 3, we can observe that both DLD-wLC and DLD-woLC outperforms the other two schemes in terms of inter-server communication whether there are hotspots or not.

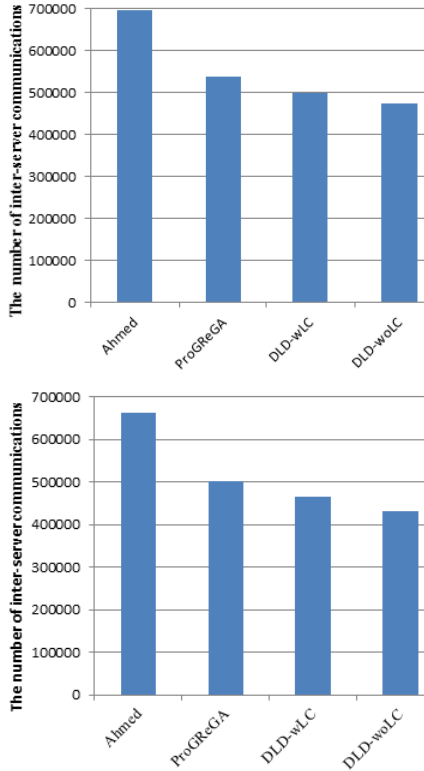


Figure 3: Inter-Server Communication Comparisons of LB Schemes with hotspots (up) or without hotspots (down).

4.2.3 Load Deviation Ratio

The Load Deviation Ratio (LDR) of a server X is defined to be the ratio of the difference between X 's load and the average load to the average load. For example, if the average load is 0.8 and X 's load is 0.85, then X 's LDR is $|0.85-0.8|/0.8=0.0625$. The LDR of a scheme is the average LDR of all servers. By Figure 4, we can observe that DLD (precisely DLD-woLC) outperforms the Ahmed and ProGReGA schemes in terms of LDR.

5. CONCLUSION

This paper proposes DLD, a cell-based dynamic load balancing scheme to achieve load balance among multiple NVE servers. DLD tries to keep servers normally-loaded and makes an overloaded server transfer its load to lightly-loaded or underloaded servers. Extensive simulation experiments are performed to compare the DLD scheme with the Ahmed and ProGReGA schemes. The proposed scheme outperforms others in terms of overheads of avatar migration, inter-server communication and load deviation ratio.

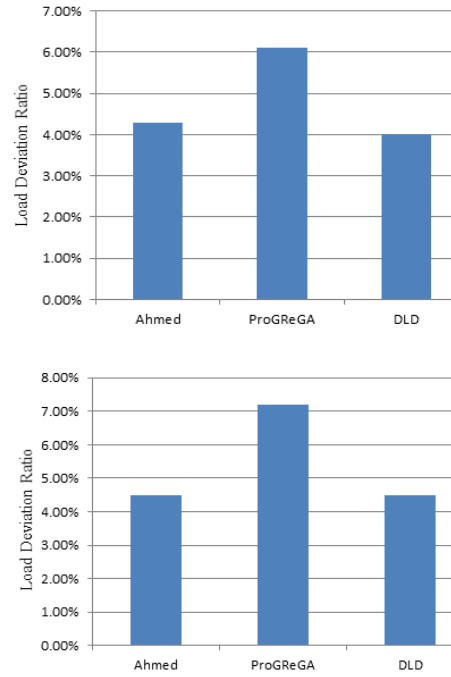


Figure 4: Load Deviation Ratio Comparisons of LB Schemes with hotspots (up) or without hotspots (down).

6. REFERENCES

- [1] S. A. Abdulazeez, A. El Rhalibi, M. Merabti and D. Al-Jumeily, "Multi-Server Dynamic Load Balancing for Networked Virtual Environments," in *Proc. of The 15th Post Graduate Network Symposium (PGNet)*, 2014.
- [2] D. T. Ahmed, and S. Shirmohammadi, "A Microcell Oriented Load Balancing Model for Collaborative Virtual Environments," in *Proc. of 2008 IEEE Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems*, pp. 86 – 91, 2008.
- [3] C. E. Bezerra and C. F. R. Geyer, "A load balancing scheme for massively multiplayer online games," *Journal of Multimedia Tools and Applications*, Vol. 45, Issue 1-3, pp. 263-289, 2009.
- [4] C. E. Bezerra, J. L. D. Comba, and C. F. R. Geyer, "A fine granularity load balancing technique for MMOG servers using a kd-tree to partition the space," in *Proc. of 2009 IEEE VIII Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, 2009.
- [5] H. Backhaus and S. Krause, "QuON: a quad-tree-based overlay protocol for distributed virtual worlds," *Int. Journal of Advanced Media and Communication*, vol. 4, no. 2, p. 126-139, 2010.
- [6] S.-Y. Hu and K.-T. Chen, "VSO: Self-Organizing Spatial Publish Subscribe," *2011 IEEE 5th Int. Conf. on Self-Adaptive and Self-Organizing Systems*, pp. 21–30, Oct. 2011.
- [7] World of Warcraft, <http://us.battle.net/wow/en>.