A Hybrid Scheme for Massive Multiplayer Online Games using Mobile Agent on Dynamic Quadtree Architecture

Wai-Tak Wong
Department of Information Management
Chung Hua University, WuFu Road,
Hsinchu, Taiwan, 30067
wtwong@mi.chu.edu.tw

Zih-Ruei Ciou
Department of Information Management
Chung Hua University, WuFu Road,
Hsinchu, Taiwan, 30067
M9110023@mi.chu.edu.tw

Yu-Cheng Chang
Department of Information Management
Chung Hua University, WuFu Road,
Hsinchu, Taiwan, 30067
M9210011@mi.chu.edu.tw

Abstract

Unlike the traditional online game infrastructure—the central server approach, we propose a hybrid scheme for multiplayer online games in this paper. This scheme effectively combines the peer-to-peer model and client-server model to solve the problems of scalability, performance and cost limitation. This scheme is suitable for online games playing in a geographical area in which the online player community is organized into quadtree architecture. A central master server is kept for login, dispatching the players and solving the trouble if there is some problems happened in the virtual world. The cost of the game servers are passed to the honored members in the community. Due to new players moving into or old players moving out of a region, a local game server loading may reach a limit, or only a few players still stay in its controlled area. Thus, a local game server may need to have a child local server to share its loading, or to be retired. In the other words, the quadtree is managed dynamically. The branches of quadtree will be extended or retracted depends on the distribution of the online players. A local game server can spawn a new local server or can be replaced by a new local server. When a new local game server is created, the game server source codes will be loaded into it. Like a mobile agent, the local game server is mobilized.

Keywords: Multiplayer Online Game; Mobile Agent; Dynamic Quadtree Architecture; Hybrid Scheme;

1. Introduction

In recent years massive multiplayer online games (MMOGS) are rapidly gaining popularity [1]. With hundreds of thousands concurrent players participate in the same online game, the software architecture faces the problem of scalability [2]. Currently, two common main game architectures in use are peer-to-peer and client/server. Peer-to-peer is a distributed, serverless architecture which increases scalability. However, quality of network connections in the internet is hardly predictable since it is not feasible to create point-to-point connections for many peers. Lack of multicast mechanism in the network will lead to a splitting of bandwidth as each message has to be sent to every player respectively. Today, most online multi-player games are implemented based on a client-server model instead of a peer-to-peer model [3]. Client/server game architectures offer a single point of game coordination. An authorized game server is set up and all players login to this game server to play a game against one another. Player actions are forwarded from each player station to the game server. Then, the game server processes the actions in sequence, and notifies player stations of the effects for their actions. In this model, the loading is on the server only. With a limited number of players, the server will handle quite well. However, it creates a bottleneck of processing and signaling when the number of players in online world increases.

A new commercial game architecture named Butterfly.net [4] is developed for hosting massive multiplayer online games. Its goal is to serve the online game players through a flexible, scalable and resilient infrastructure will grow as needed. In the Butterfly Grid architecture all servers are fully meshed that means each server is connected to all others over high-speed fiber-optic lines. In the course of a game, each server will communicate- or multicast-to all other servers in the grid in real time. Under this “peer-to-peer” networking approach, players are transparently routed to the optimal server in the

Grid such that server resources are allocated to the most popular games. Though this game infrastructure can eliminate the bottlenecks, provide reliability, and gain many advantages, the cost of the architecture is extraordinarily high and tightly coupled to business model.

To solve the increasing complexity of the interactions between massive game players, many researchers proposed an approach to disperse players based on geography and population for high performance systems. Benford and Fahlén [5] proposed a spatial model of interaction which had influenced the DIVE [6] and Virtual Society [7] systems. Based on this spatial model and awareness-based communication among users, Greenhalgh [8] developed the MASSIVE-1 and MASSIVE-2 distributed virtual environment system. Macedonia [9] proposed to restrict the number of interactions by coarse-grain partitioning the world into a grid of disjointed hexagonal cells For each a multicast address is associated statically. To reduce the number of simultaneous allocated multicast addresses and to optimize cell sizes, Léty [10] introduced dynamic cells. However, they didn't have a well dynamic structure to manage those cells. In this paper, a flexible and extensible infrastructure for MMOGS is proposed. The framework is a hybrid scheme of peer-to-peer model and client-server model. The master server will assign players to different local servers based on their geographic entrance to the game. The virtual world is managed by using dynamic quadtree structure depending on the population distribution of the players in each region. Regions can be merged or split according to the quadtree property. Thus, the whole player community is managed under a dynamic quadtree structure, which is to be extended or retracted from time to time depending on the population distribution. The rest of the paper starts with a description of game infrastructure in quadtree architecture. The modeling for the population distribution in dynamic quadtree architecture is illustrated in Section 3. Some critical issues of the proposed hybrid scheme are discussed in Section 4 following by some conclusions are made in Section 5.

2. The Proposed Game Infrastructure in Quadtree Architecture

Quadtree is a well-known representation of the hierarchical data structures in the fields of computer vision, computer graphics, image processing, cartography, and geographic information system [11]. The corresponding quadtree of an image is constructed by successively sub-dividing the image into four equal-size sub-images in the *NW* (northwest), *NE* (northeast), *SW* (southwest), and *SE* (southeast) quadrants. A heterogeneously colored quadrant of the image is represented by an internal node and is further divided until each sub quadrant is in the same color. The leaf node with a black (or white) color is called the black (or white) node and the internal node is called the gray node. A quadtree representation of an image is shown in Fig. 1. To use quadtree for population distribution, a black node can represent that there are players in the quadrant. If no player is in a quadrant, we use a white node to represent it. If a quadrant has a high population of players that a single machine cannot handle it, then we will subdivide the quadrant into four equal-size sub quadrants. If all four sub quadrants are in black color (has players in its quadrant), then such sub quadrants are called "virtual" leaf nodes, which have been widely used in image processing [12]. In short, the quadtree representation possesses several desirable features that make it particularly suitable for applications. For example, its hierarchical structure makes it scalable and flexible.

As aforementioned, if one wants to implement a massive multiplayer game, a huge map with a huge number of players has to be implemented. To handle this, our strategy is to split the overall communication bandwidth into smaller pieces and to distribute the computational loading over many machines instead of one single server.

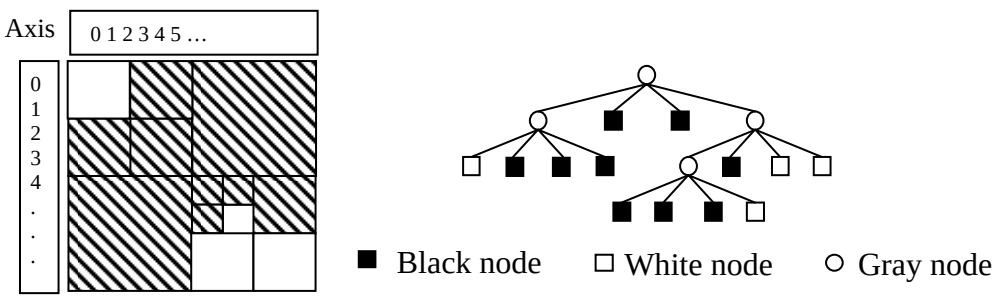

Fig. 1. An image and its quadtree representation

Fiedler [13] proposed a topological division of the map into smaller pieces. They tessellated the map into pieces of the magnitude of normal multiplayer maps. They implemented two map types, rectangular tile and hexagonal tile. However, both map types did not have the hierarchy to divide and conquer the population distribution. The fundamental concept of the tessellation for the players is that a single player needs not to know every thing on the map if it does not affect him. If any player is beyond the scope of his visibility (his active area), he needs not to have any notion of them. In this paper, quadtree structure is used to tessellate the online players' distribution in the virtual world. The whole quadtree structure of the virtual world is managed in the central master server. The proposed game architecture in quadtree structure is shown in Fig. 2 where the host game server is the first game server. In this scheme the number of players is restricted in each local game server. However, the controlled geographic area under a local game server may be big. Therefore, a quadtree structure is maintained in the local game server to speed up the interaction among its players. Here, we name the controlled area as scene region. Based on the player distribution the scene region may be further subdivided according to the quadtree property. The basic unit of a quadtree, a leaf node at the bottom level, can be defined as some reasonable size of area (or a room). If a room is located at the boundary, we name it boundary room. If a room is used as an entrance for the new players, we name it entrance room. A scene region can have more than one entrance. For all type of rooms, their sizes are equal. The room size must be larger than the player's sphere of *influence* (SOI), which was proposed by Baughman [3]. The *influence* is defined as any in-game information that affects a player's decisions, where in-game refers to parts of the game world, as opposed to external knowledge that the player may have, e.g., that a certain opponent typically follows some strategy. To avoid a large amount of ineffective information exchanging between a player and other players who locate far beyond its SOI, the publisher/subscriber model of communication is adopted in this system. Each room is managed as a channel in the local game server. The quadtree structure maintained in a local game server is depicted in Fig. 3. If a player registers to one room, it will subscribe the corresponding room channel and its eight neighbors' channels. As shown in Fig. 4, the eight neighbors of R_{a4} are R_{a5} (east), R_{c1} (south), R_{a3} (west), R_{a1} (north), R_{a0} (northwest corner), R_{a2} (northeast corner), R_{c0} (southwest corner) and R_{c2} (southeast corner), respectively. When a player migrates from one room to another, the non-neighbors' room channels are unsubscribed and the new neighbors' room channels are subscribed. The variation of neighborhood during the migration of a player from one room to another room is also described in Fig. 4. If a player enters a boundary room of a scene region, e.g. R_{a4} of *A*, it will subscribe another boundary rooms which belongs to its neighbor scene regions, e.g. R_{c0}, R_{c1}, R_{c2} of *C*.

3. Modeling in Dynamic Quadtree Architecture

During the online game is running we have to consider the following scenarios for the proposed dynamic quadtree architecture.

3.1 When a player enters a scene region

If a player, named *X*, wants to play the online game based on our proposed architecture, player *X* have to go to the master server to login to his account. After authentication the master server will transfer the account information to player *X*. Play *X* will need to choose an entrance, says *Y*, in the game map to start the game. Then, the master server will notify player *X* where the local server is. Let's assume that local game server *Z* will serve the game for player *X*. At the same time, the master will notify local game server *Z* that player *X* (with the account information) is coming from entrance *Y* soon. When player *X* joins local game server *Z*, the validness of the account information will be checked. If there is no problem, player *X* need to subscribe the channel of entrance room *Y* and the channel of the eight neighbors of room *Y*. The whole process is depicted in Fig. 5.

3.2 When a player migrates from one scene region to other region

As shown in Fig. 4, when the player approaches the border of a scene region *A*, he also needs to know what happens in the adjacent scene regions *B*, *C* and *D*. If a player crosses a boundary to other scene region, let say from *A* to *C*, the local server of *A* must notify and pass the information of the player to the local server of *C*. The local server of *A* also notifies the player that he must join the local server of *C* and pass his account information to it. When the local server of *C* receives the account information, it will verify the correctness. If there is conflict, the confliction will be sent to the master server for further judgment. If it is correct, then the player will unsubscribe channel R_{a0}, R_{a1}, R_{a2}, R_{a3}, and R_{c0}, and subscribe channel R_{c3}, R_{c4}, R_{b0}, R_{d0}, and R_{d1}. Finally, the player will continue to play the online game in scene region *C*. The whole process is depicted in Fig. 6. The subscription

and un-subscription of a room channel are automatically detected and performed. The action of un-subscription will be delayed for a short period of time to avoid the player returning back immediately.

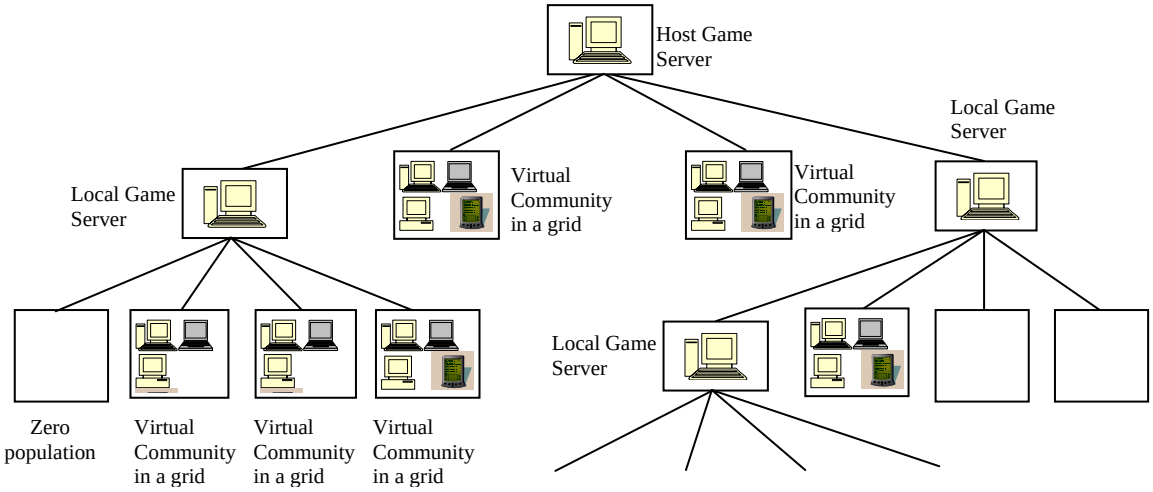

Fig. 2. The proposed game architecture corresponding to the quadtree in Fig. 1.

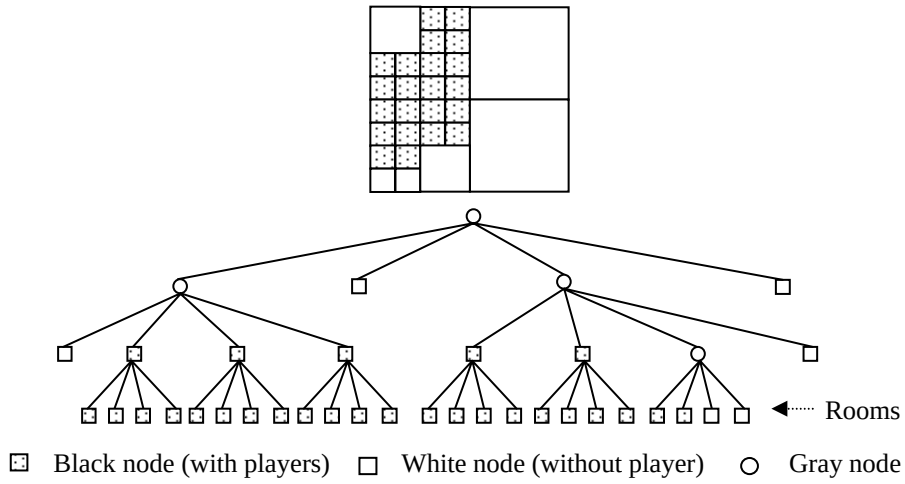

Fig. 3. The quadtree structure maintained in a local game server.

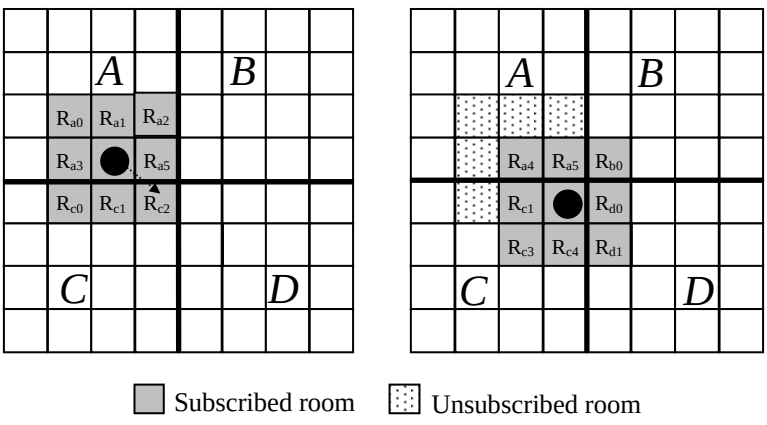

Fig. 4. The variations of neighbors during the migration of a player from R_{a4} to R_{c2}.

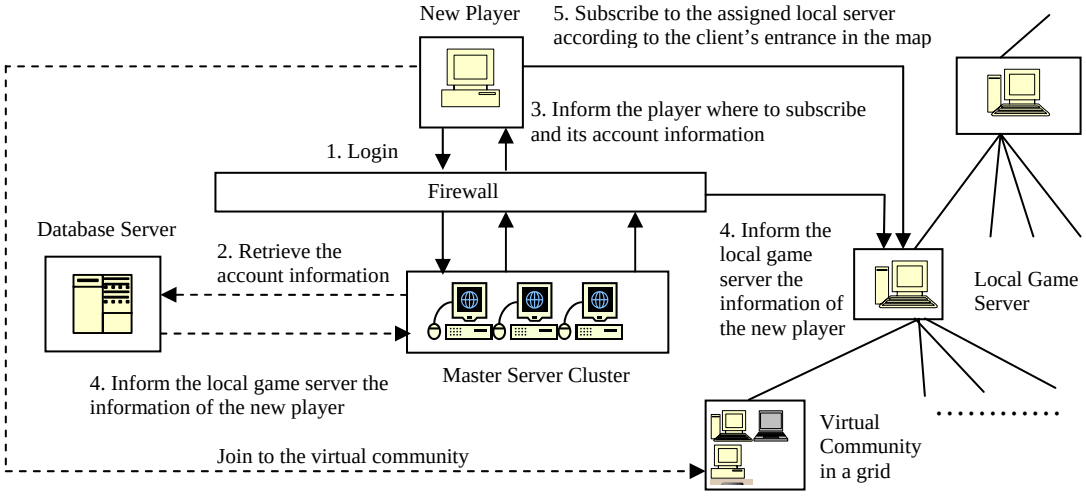

Fig. 5. The steps need to perform when a player enters a scene region.

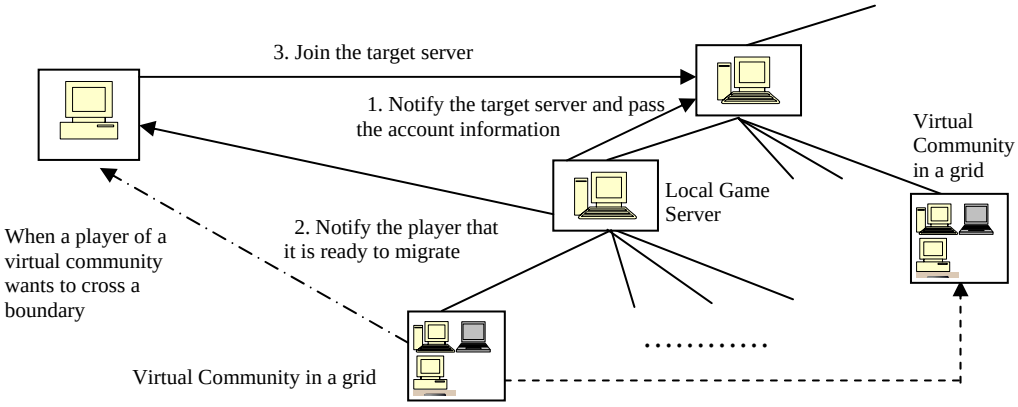

Fig. 6. The steps need to perform when a player migrates from a scene region.

3.3 When a player wants to quit the game

When a player wants to quit the game, he must not kill the application or shutdown the computer. Otherwise, the master server will think that the client has left abruptly. The client's credit may be reduced for this reason. The normal procedure is to quit the application. The application will notify the local server and the local server will pass the player's account information to the master server. The master server will temporary save the information and wait for the player to make sure the correctness of the account information. The application will also send the player's account information to the master server. If there is no difference between both records, the database will be updated. Otherwise, the master server must find out who produces the trouble. Currently the troubleshooting procedure is being investigated. The whole process is depicted in Fig. 7.

3.4 When a local server wants to quit

When a player wants to quit the game and his role is also a local game server, he needs to perform a number of steps. The first step is that the local server must find out a candidate from all online players under his control to replace his role. Then, the local server must notify the master server that he wants to quit and who the candidate is. If it is ok, the master server will validate the candidate, make it to a new local server and also update the quadtree structure. The old local server will pass the account information of all its players to the new local server and then notify them to switch to the new local server. Then, the players will connect to the new local server and pass their account information for validation. If there is no problem, the players will continue the game. If there is something wrong, the local server will pass the problem to the master server. The whole process is depicted in Fig. 8.

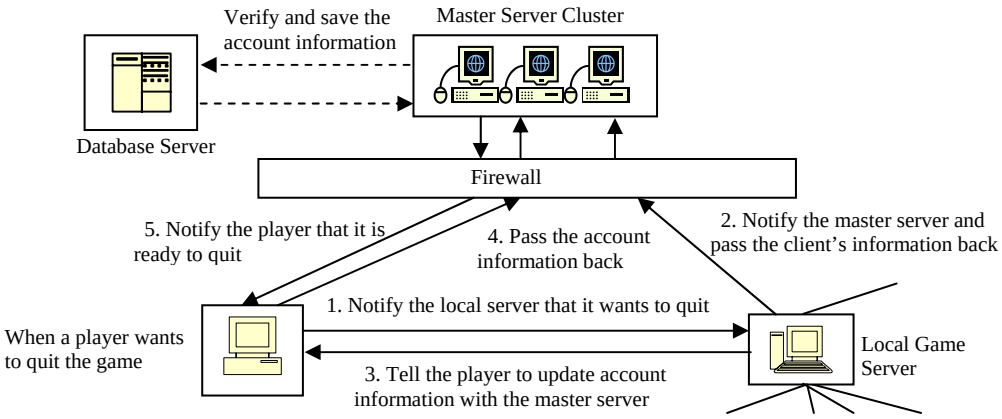

Fig. 7. The steps need to perform when a player want to quit the game.

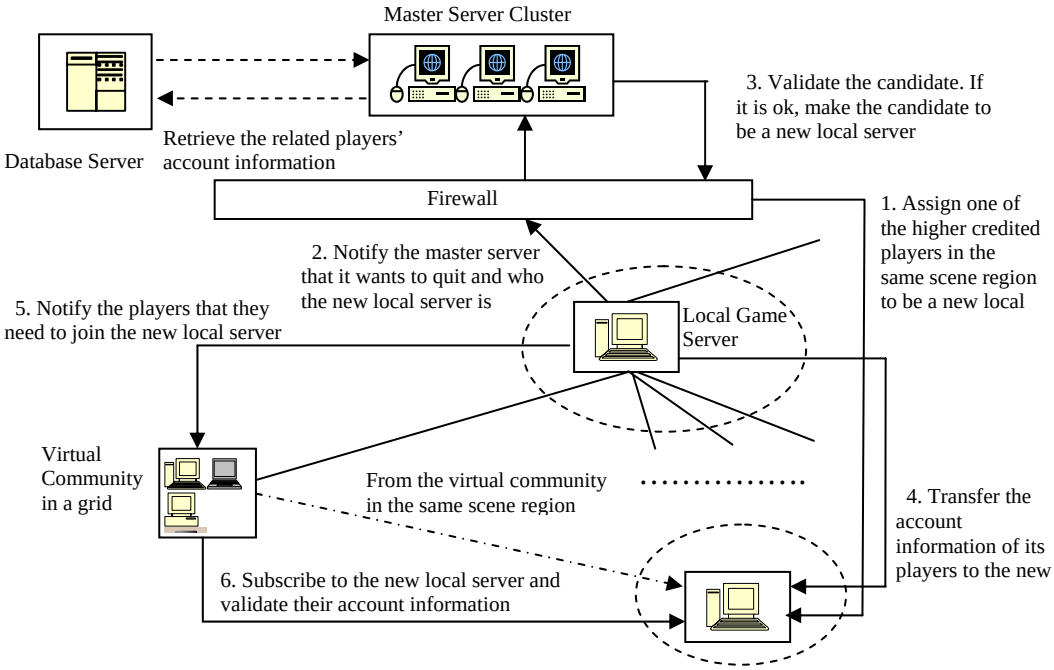

Fig. 8. The steps need to perform when a local game server wants to quit the game.

3.5 When a local server quits abruptly

If a player finds that the communication of its local server is broken, it will try to reconnect to the local server and to restart from its last status. It will also report the error to the master server for statistical evaluation. If the reconnection to the local server gets failure, it will report the death of the local server to the master server. The master server will confirm the death by communicating to the local server. If it gets failure to connect, then it thinks that the local server is dead. Thus, the local server will find out a suitable online player (with highest credit) and crowned it to be a local server. All the players who participates the game in such region will be rolled back to the states according to their local account information. Then everyone start playing the game again. In this scheme, there is no need for heart breaking detection to the local servers. It can reduce the loading of the master server. The whole process is depicted in Fig. 9.

3.6 When a player quits abruptly

If a local server detected that the communication of a player is broken, then it will notify the death of the player with its account information to the master server. The master server will wait for the player for a short period of time. Based on our mechanism, if the connection between a player and a local server is broken, the player will report to the master server. If the player does not report to the master server, then he will be counted as dead. The master server will use the account information from the local server to update database. If the master server receives the connection from the player, the master server will try to find out what the problem is. If it

knows who causes the trouble, somebody's credit may be reduced. The whole process is depicted in Fig.10.

3.7 Other issues

When the local server fully loads, it will find a helper to share its loading. Thus, the quadtree maintained in the master server will be extended. When the local server's loading becomes low, it will ask for its parent local server to take over its work. Thus, the quadtree maintained in the master server will be retracted.

4. Discussion

4.1 About Authentication Problem

As aforementioned, when a player wants to quit from the game, both the player and local server must send the last game state to the master server. Then the master server will compare these two copies to make sure those data are identical**.** To reduce the effort of sending game state, the local server may send the full game state information to the master server while the client sends the digital digest. The master server can generate a digital digest from the data sent by the local server and compares it to the digest coming from the player. Since digital digest of game state is much smaller to the original data, the bandwidth usages of the master server and the player are reduced.

4.2 Credit

In our approach, game players are allowed to host the game voluntarily. Therefore, we need an effective way to find who are suitable to be the local game servers. The hardware, the network bandwidth and the total participation time of the players will be the criteria to build the credit system.

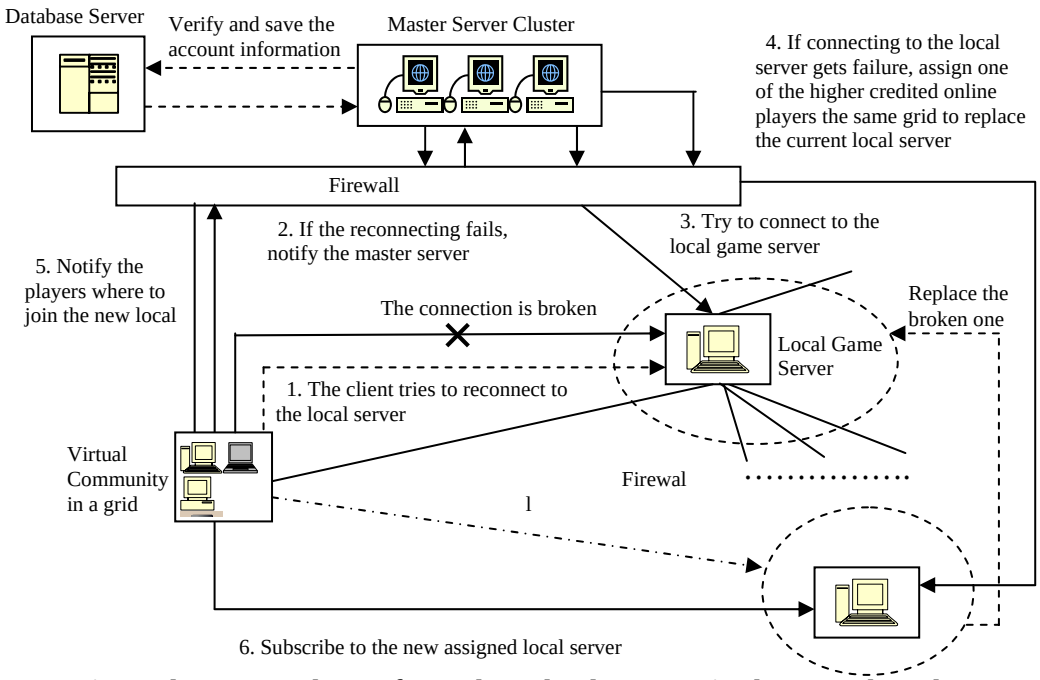

Fig. 9. The steps need to perform when a local server quits the game abruptly.

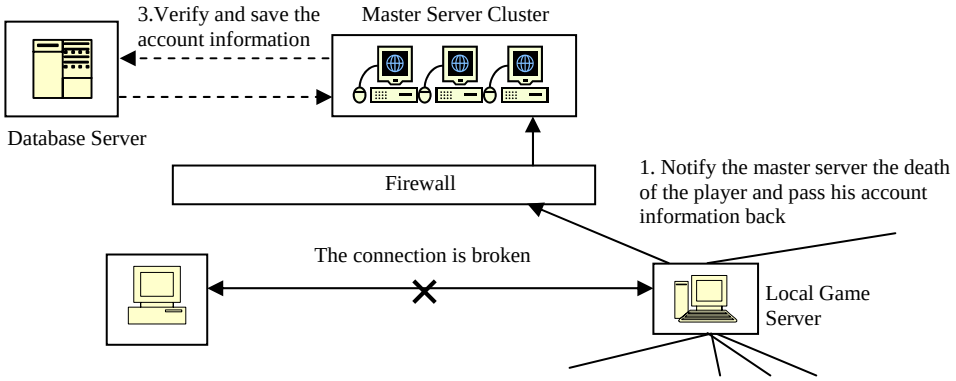

Fig. 10. The steps need to perform when a player quits the game abruptly.

4.3 Cheat-Proof

Fairness is one of the important issues in the game design. Under our distributed architecture, we must prevent cheating from the player and also the local server. By comparing the two game states, we can validate the correctness of the game state. However, if both sides cheat in co-operation, we still have no effective way to prevent it right now. Further research will focus on the improvement for the fairness of the game.

5. Conclusion

A hybrid scheme using mobile agent on dynamic quadtree architecture for massive multiplayer online games is presented in this paper. Our proposed architecture is effective in distributing game players to multiple servers. It is easy to split the game map to smaller regions to increase the participated local servers when there are too many players. In contrast, regions with only a few players in it will be merged together based on quadtree structure. With this ability for load balancing plus an effective authentication mechanism, we could allow game players to share the loading voluntarily by hosting the game. This type of cooperation will significantly reduce the operation cost of an online game. The proposed distributed architecture provides important features including fault tolerance and scalability. Unlike the traditional client-server architecture, everyone is influenced when the server crashes. In addition, our scheme is simply a client-server architecture under a local server. Therefore any online game which is currently running on traditional client-server architecture can be easily ported to ours without any risk of rebuilding program infrastructure.

References

[1] S-J. Kim, F. Kuester and K.H. Kim, “A global timestamp-based scalable framework for multi-player online games,” *Proceedings of the IEEE Fourth International Symposium on Multimedia Software Engineering*, Newport Beach, California, USA, pp. 2-10, 2002.

[2] A.R. Bharambe, S. Rao and S. Seshan, “Mecury: a scalable publish-subscribe system for internet games”, *Proceedings of the first workshop on Network and system support for games*, Braunschweig, Germany, pp. 3–9, 2002.

[3] N.E. Baughman and B.N. Levine, “Cheat-Proof playout for centralized and distributed online games,” *Proceedings of the Twentieth IEEE Computer and Communication Society INFOCOM Conference*, Alaska, USA, pp.104-113, 2001.

[4] Butterfly.net: Powering Next-Generation Gaming with Computing on Demand http://www.butterfly.net/platform/technology/idc.pdf

[5] S.D. Benford and L.E. Fahlén, “A spatial model of interaction in large virtual environments,” *Proceedings of the Third European Conference on Computer-Supported Cooperative Work - ECSCW '93*, Milano, Italy, pp.107-132, 1993.

[6] C. Carlsson and O. Hagsand, “DIVE – A multi-user virtual reality system,” *IEEE Virtual Reality Annual International Symposium*, Washington, USA, pp. 394-400, 1993.

[7] R. Lea, Y. Honda, and K. Matsuda, “Virtual Society: Collaboration in 3D Spaces on the Internet,” *Computer Supported Cooperative Work : The Journal of Collaborative Computing*, vol. 6. pp. 227-250, 1997.

[8] C. Greenhalgh, “Awareness-based communication management in the MASSIVE systems,” Distributed System Engineering, pp. 129-137, 1998.

[9] M.R. Macedonia, M.J. Zyda D.R. Pratt, D.P. Brutzman and P.T. Barham, “Exploiting reality with multicast groups,” *IEEE Computer Graphics Application*, vol. 15, pp.38-45, 1995.

[10] E. Léty, *Une Architecture de Communication pour Environments Virtuels Distribués á Grande-Échelle sur l’Internet*, PhD thesis, Université de Nice-Sophia Antipolis, Decemeber 2000.

[11] H. Samet, *Applications of Spatial Data Structures – Computer Graphics, Image Processing, and GIS*, Addison-Wesley, New York, 1990.

[12] F. Y. Shih and W. T. Wong, ‘‘An adaptive algorithm for conversion from quadtree to chain codes’’, *Pattern Recognition*, vol. 34, pp. 631-639, 2001.

[13] S. Fiedler, M. Wallner and M. Weber, communication architecture for massive multiplayer games,” *Netgames 2002 first workshop on network and system support for games*, Braunschweig, Germany, pp. 14–22, 2002.