An Efficient Hybrid P2P MMOG Cloud Architecture for Dynamic Load Management

Ginhung Wang, Kuochen Wang

Abstract- In recent years, massively multiplayer online games (MMOGs) become more and more popular. We combine cloud computing with MMOGs to increase flexibility of resource allocation. In an MMOG cloud environment, we use virtual machines (VMs), instead of traditional physical game servers. A game world is divided into several game regions. Each game region is serviced by at least one VM. Peer-to-peer (P2P) cloud computing is a new approach that combines high computation power, scalability, reliability and efficient information sharing of servers. This paper proposes a hybrid P2P cloud architecture for MMOGs which includes two-level load management, multi-threshold load management for each game server and load management among game servers. It is suitable for players to interact with P2P cloud servers and it avoids bottlenecks of the current multi-server MMOG architecture. Simulation results show that the proposed hybrid P2P cloud architecture can reduce the average response time by 20.6% compared to the multi-server architecture under medium to high load through flexible allocation of resources (virtual machines).

Key words: cloud computing, load management, massively multiplayer online game, peer-to-peer, resource allocation.

I. INTRODUCTION

MMOGs have emerged in the past decade as a new type of large-scale distributed applications. An MMOG is a real-time virtual world with many players across the real world. Traditionally, MMOGs operate as multi-server architecture which is composed of many game servers. Game servers simulate a virtual game world. They receive and process commands from clients (players), and inter-operate with a billing and accounting system [1]. To support ever more and more players, MMOG environments require a new architecture with a high degree of flexibility to respond to environmental changes, including load change. Therefore, in this paper, we propose a hybrid P2P cloud architecture for MMOGs to provide flexible resource usages to deal with load change.

The game providers need to install and operate a large infrastructure, with hundreds to thousands of computers for each game [2]. For example, the operating infrastructure of an MMOG, World of Warcraft, has over 10,000 computers [3]. The traditional solutions to decrease server loads are deploying more servers to the heavy-loaded areas [2]. This solution is simple but has limitation of scalability. The other solution is using the multi-server architecture, where game servers can be added to heavy-loaded areas on demand. In this paper, we propose a hybrid P2P cloud architecture for MMOGs which includes two-level load management: multi-threshold load management for each game server and load management among game servers to resolve current architecture drawbacks.

II. RELATED WORK

A. Client-server MMOG architecture

Client-server MMOG architecture has players (clients) that send requests to a single server. It is simple and easy to implement, but it is less scalable. As shown in Figure 1, commands from players must go through a single server, and the single server handles commands and returns state updates to players. The single server must be in the presence of insufficient bandwidth and copious waiting time for players. To resolve the problems, there are several approaches [13] to distribute traffic among multiple servers, such as a mirrored server [14], generic proxy [15], or booster box [16].

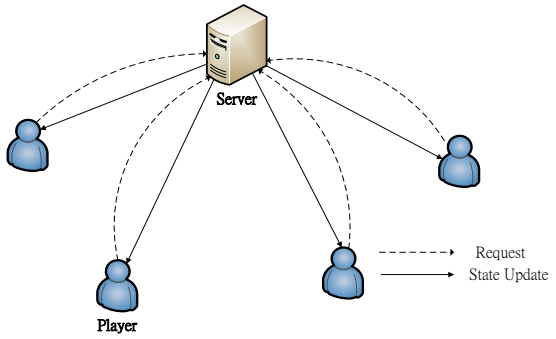

Figure 1: Client-server MMOG architecture [28].

B. Multi-server MMOG architecture

Multi-server MMOG architecture [5] has various kinds of servers which provide different functionalities as shown in Figure 2. The various servers are described as follows [5]

Master Server handles access of players and communicates with operated servers. It assists players logging in a zone server and transfers data of players to the corresponding zone server from the character server. Character Server is mainly stored the data of players. It assists players in using the same characters on any zone servers. World Daemon transfers players from a zone server to another zone server that players expected. It also monitors each load of zone servers. Zone Cluster Server is composed of zone servers and mainly serves players. A zone server is assigned to serve a specific zone. A client moves

between zone servers. A client sends a request to an interested zone server.

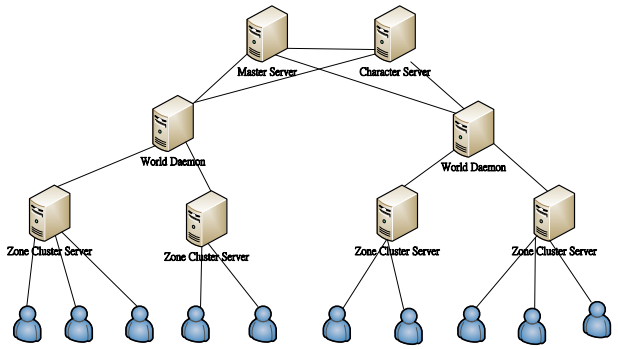

Figure 2: MMOG multi-server architecture [5]

In this MMOG multi-server architecture, the players in the same zone connect to the specific zone server. Some zone servers will suffer heavy load if many players move into the same zone. Hence, load balancing is needed to prevent the downgrade of service. In [10], it creates a management server that monitors zone servers and decides a load balancing strategy. In this case, the management server would become the bottleneck of the system [8].

C. P2P cloud computing architecture

As shown in Figure 3 [12], the architecture includes three basic roles: User, Central Peer and Side Peer. The Central Peer and Side Peer form two P2P networks, called central P2P network and side P2P network, respectively. The central P2P network mainly maintains metadata of dynamic mapreduce and backups DHT P2P storage cloud. The side P2P network is mainly used to provide storage and computing resources.

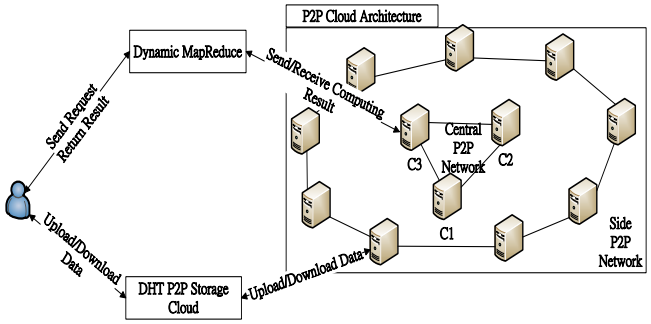

Figure 3: Cloud computing architecture based on P2P [12]

III. PROPOSED HYBRID P2P MMOG CLOUD ARCHITECTURE

A. Proposed cloud architecture

MMOG environments require a high degree of flexibility to respond to environmental changes, including load change. Thus, we combine cloud computing with MMOGs to increase flexibility of resource allocation. The proposed cloud architecture includes four basic components: game server cloud, players, character database and game regions data storage, as shown in Figure 4. We introduce each component's characteristics as follows:

Game servers cloud consists a number of game servers and a game server has a number of virtual machines (VMs) to serve players. A game server receives requests from players, calculates new game states in regions, and responses to players. A game server is also in charge of creating accounts for players when they login for the first time. Players move between game regions of a game world. The load of a game region increases when players move into the game region. Character database stores the data of players, such as equipment, rank, site, etc. The backup method is based on backup method of hadoop. Game regions data storage maintains game states in each region. It also backups game states and object states in each game region.

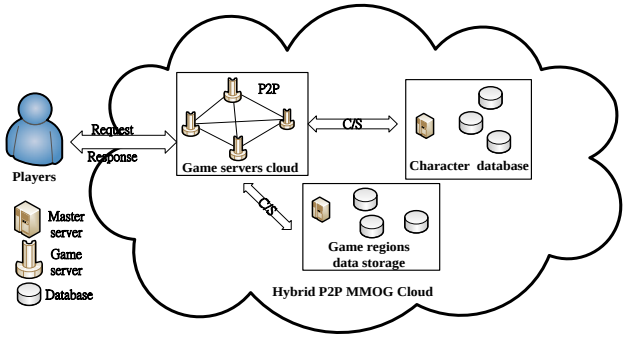

Figure 4: Hybrid P2P MMOG Cloud architecture

B. Game server cloud

In MMOG environments, there are huge messages, especially communication messages between players. Game servers use a P2P protocol to exchange load information and game region data. Game servers also use the P2P protocol to exchange the information of players. By P2P flooding, each game server can know the other game servers' load. When a game server is overloaded, it will migrate some players to other game servers by the proposed game server load management procedure.

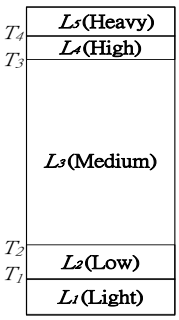

Figure 5: Multi-threshold load management

C. P2P Cloud Computing Load management

In the game servers cloud, we allocate VMs from game servers to service game regions. Game regions do not setup on specific virtual machines. In this way, it could help the system to get high scalability. We do not use a single master server in the game servers cloud, Game servers communication is based on a P2P protocol. Game servers can exchange load information by flooding in the P2P protocol. Game servers can perform load management by themselves, and a game server is also in charge

of creating accounts for players when they login for the first time. This way can help resource allocation more efficient.

The multi-threshold load management focuses on VMs management. We use four thresholds (T_1, T_2, T_3, and T_4) to define five loading layers (L_1, L_2, L_3, L_4, and L_5) for each VM. T_1 is the lower bound of VM load. T_2 and T_3 define two preferred loading layers to service players. T_4 is the upper bound of VM load. As shown in Figure 5, each loading layer represents a different load and status.

- L_1 (Light): For a VM_i operating in this layer, its players will be migrated to another VM_j serving the same region in L_3, L_2, or L_4 (selection order). After the players in VM_i are migrated out, VM_i will be released.
- L_2 (High): For a VM_i with maximal load in this layer, it can provide capacity for a VM_j in L_5 to migrate its players to this VM_i
- L_3 (Medium): It is the optimal layer for VMs to stay.
- L_4 (Low): For a VM_i operating in this layer, its loading is slightly high, but is tolerable.
- L_5 (Light): For a VM_i operating in this layer, part of its players will be migrated to another VM_j serving the same region in L_2, L_3, or L_1 (selection order) if there are available VMs in these layers; If there are no VMs in these layers, the game server load management will be executed.

Figure 6 and Figure 7 show the flow chart of the multi-threshold load management algorithm. The details of the multi-threshold load management algorithm are described in the following.

1) We monitor the VMs in each game region and sort the VMs in each game region according to their loadings in an ascending order.
2) If the VM in L_5 is empty, go to step 7. Otherwise, go to step 3.

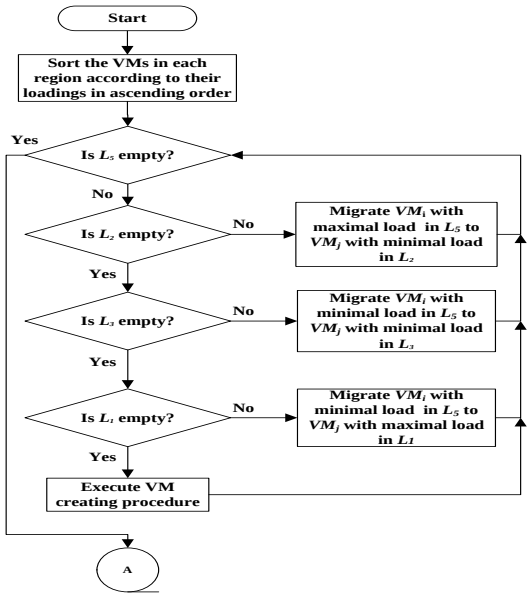

Figure 6: Multi-threshold local load management algorithm (1/2)

3) In this step, we know some VMs exist in L_5. We check if there are VMs in L_2. If there are VMs in L_2, we migrate VMi with maximal load in L_5 to the VM_j with minimal load in L_2, and then go to step 2. Otherwise, go to step 4.
4) We check if there are VMs in L_3. If there are VMs in L_3, we migrate VM_i with minimal load in L5 to the VM_j with minimal load in L_3, and then go to step 2. Otherwise, go to step 5.
5) We check if there are VMs in L_1. If there are VMs in L_1, we migrate VM_i with minimal load in L_5 to the VM_j with minimal load in L_1, and then go to step 2. Otherwise, go to step 6.
6) Executing the VM creating procedure to create a VM to share the load in this game region.
7) In this step, we start to release VMs. Check if there is any VM in L_1. If there is no VM in L_1, exit this procedure. Otherwise, go to step 8.
8) We check if there are VMs in L_3. If there are VMs in L_3, we migrate VM_i with minimal load in L_1 to the VM_j with minimal load in L_3, release VM_i, and then go to step 7. Otherwise, go to step 9.
9) We check if there are VMs in L_2. If there are VMs in L_2, we migrate VM_i with minimal load in L_1 to the VM_j with maximal load in L_2, release VM_i, and then go to step 7. Otherwise, go to step 10.
10) We check if there are VMs in L_4. If there are VMs in L_4, we migrate VM_i with minimal load in L_1 to the VM_j with maximal load in L_4, release VM_i, and then go to step 7. Otherwise, exit the procedure.

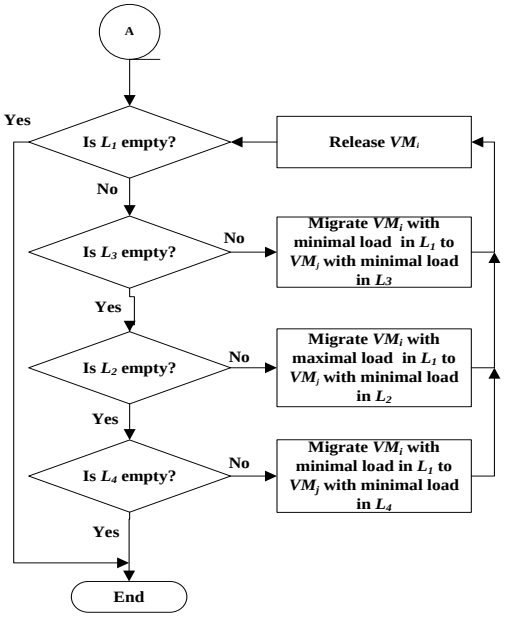

Figure 7: Multi-threshold local load management algorith (2/2)

When a game region needs a new VM, the VM creating procedure is executed, as shown in Figure 8. Firstly, the game server checks its resource. If it has enough resources, the game server will create a new VM to service the game region. If not, the game server will turn down the VM creating request. The game server executes the game server load management procedure to transfer players to another game server.

In the game server load management procedure, we mainly consider the physical distance between players and game servers, because players need to be served with low response time. As shown in Figure 9, firstly we find a neighboring game server who is serving the same game region and has the lowest load.

If not available, we select a neighboring game server that is not serving the same game region and has the lowest load. If not available, we select a distant game server that has capacity to service more players. The selected game server will service the migrated players. Finally, if we could not find a game server who has capacity to service more players, then it means all game servers are overloading.

IV. SIMULATION

A. Simulation setup and evaluation metrics

We use CloudSim [12] to simulate players' behaviors by sending service requests to game servers. Players' requests were collected from Stendhal MMORPG using Wireshark. The Stendhal MMORPG is running on Intel i7 2.93 GHz CPU with 512 MB RAM and 100 Mbps Ethernet. The average service time to serve a request is 0.2 *ms*. There are three game servers and the upper bound of the response time is set to 300 *ms* according to [11]. Each game server can create at most 40 VMs. The capability of a VM is 2000 MIPS with 1024 MB RAM and 100 Mbps Ethernet (P3 capability). Each VM can support at most 59 players (obtained from CloudSim). The total number of game regions is 30. In addition, a game region is served by at least one VM, if there are players.

The *average response time* is defined as the elapsed time between the time a player sends a request to the time that a player actually receives the response, which is formulized as follows:

$$T_{avg} = \frac{1}{mp}\sum_{k=1}^{m}\sum_{j=1}^{p}\sum_{i=1}^{n_j}\frac{TR_i - TS_i}{n_j}$$

where T_{avg} is average response time, TR_i is the time interval for player to receive a response, and i is the i^{th} request. TS_i is the time interval for player j to send a request, and i is the i^{th} request. p is the total number of player j. n_j is the total number of requests for player j. k is the k^{th} round, and m is the total rounds of execution.

The average *deadline miss ratio* is defined as the response time being more than a real time bound, and we set the real time bound to 300 *ms* according to [11], which is formulized as follows:

$$\mathrm{D_{avg}} = \frac{\sum_{j=1}^{m}\frac{RD_j}{RT_j}}{m}$$

where D_{avg} is deadline miss ratio, RD_j is the total number of responses that miss the deadline in a game server j, RT_j is the total number of responses in a game server j, j is the j^{th} game server, and m is total number of game servers

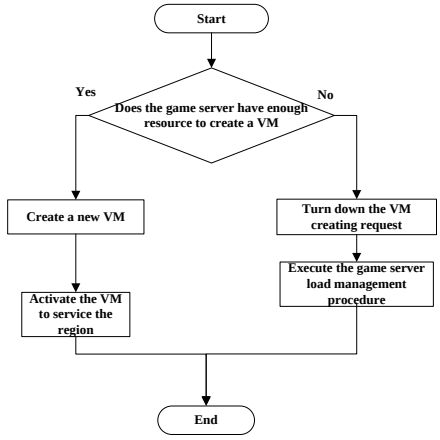

Figure 8: VM creating procedure

B. Comparison between multi-server and hybrid P2P cloud

Figure 10 shows the average response time of the multi-server architecture and proposed hybrid P2P cloud architecture. The multi-server architecture has the lower response time initially because a regional server's capacity is higher than a VM's. When the number of players increases to 3490, the multi-server architecture has higher response time. This is because its regional servers serve specific game regions. As a result, players are queued up for a specific regional server. In our architecture, each VM has lower capacity, but our architecture has higher scalability. VMs can be migrated to a busy game region and the response time of players can be reduced. Because of this, our architecture performs better after the number of players exceeding 3490.

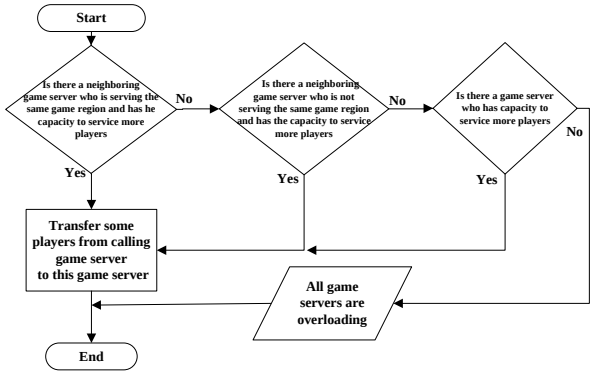

Figure 9: Game server load management procedure

Figure 11 shows the deadline miss ratio between the multi-server architecture and proposed hybrid P2P cloud architecture. In the multi-server architecture, it has the lower deadline miss ratio under lower number of players. It shows that with enough resources in the multi-server architecture, the multi-server architecture can have good performance. The proposed architecture has a smaller deadline miss ratio under medium to high load than the multi-server architecture.

V. Conclusion

A. *Concluding remarks*

In this paper, we propose a hybrid P2P cloud architecture which includes multi-threshold load management and game server load management for MMOG cloud environments. The multi-threshold load management can make the utilization of virtual machines more efficient. The game server load management transfers some players from an overloaded game server to a neighbor game server with available capacity. The proposed hybrid P2P cloud architecture can support 10.31% more players under no deadline (300 ms) miss compared to the multi-server architecture. The proposed hybrid P2P cloud architecture can reduce the average response time by 20.6% compared to the multi-server architecture under medium to high load through flexible allocation of resources (virtual machines). The proposed architecture also has a 27.9% smaller deadline miss ratio than the multi-server architecture under medium to high load.

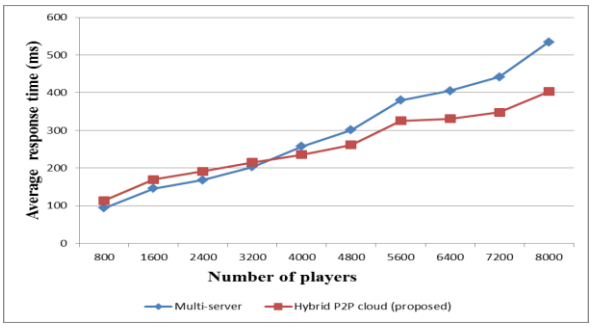

Figure 10: Average response time between multi-server and proposed hybrid P2P cloud architecture.

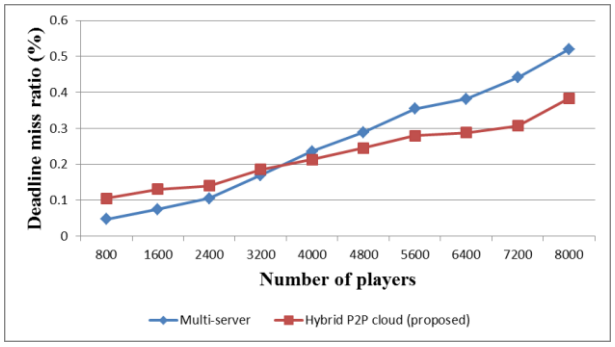

Figure 11: Deadline miss ratio between multi-server and proposed hybrid P2P cloud architecture.

VI. References

[1] A. Shaikh, S. Sahu, M.-C. Rosu, M. Shea, and D. Saha, “On demand platform for online games,” IBM Systems Journal, vol. 45, no. 1, pp. 7–20, 2006.

[2] B. Hack, M. Morhaime, J.-F. Grollemund, and N. Bradford, “Introduction to vivendi games,” [Online] Available: http://www.vivendi.com/, Jun 2006.

[3] S. Sukhyun, et al., "Blue Eyes: Scalable and reliable system management for cloud computing," in Proceedings of the IEEE International Symposium on Parallel & Distributed Processing, pp. 1-8, 2009.

[4] V. Nae, A. Iosup, and R. Prodan, "Dynamic resource provisioning in massively multiplayer online games," IEEE Transactions on Parallel and Distributed Systems, vol. PP, pp. 1-1, 2010.

[5] Torque MMO Kit - Server Architecture http://www.mmoworkshop.com/trac/mom/wiki/ServerArchitecture

[6] A. E. Rhalibi and M. Merabti, "Interest management and scalability issues in P2P MMOG," in Proceedings of the Consumer Communications and Networking Conference, pp. 1188-1192, 2006.

[7] X.-B. Shi, D. Yang, L. Du, and Z.-W. Wang, "Research on service management techniques for P2P MMOG," in Proceedings of the International Conference on Internet Technology and Applications, pp. 1-4, 2010.

[8] P. Zhao, T.-l. Huang, C.-x. Liu, and X. Wang, "Research of P2P architecture based on cloud computing," in Proceedings of the International Conference on Intelligent Computing and Integrated Systems, pp. 652-655, 2010.

[9] C. Carter, A. E. Rhalibi, M. Merabti, and A. T. Bendiab, "Hybrid client-server, peer-to-peer framework for MMOG," in Proceedings of IEEE International Conference on Multimedia and Expo, pp. 1558-1563, 2010.

[10] Q. Zhaoyang and W. Yanguang, "The design of the substation simulation model of distributed virtual environment," in Proceedings of the Second International Symposium on Computational Intelligence and Design, pp. 249-252, 2009.

[11] B. Hariri, S. Shirmohammadi, and M. R. Pakravan, "A distributed topology control algorithm for P2P based simulations," in Proceedings of the Distributed Simulation and Real-Time Applications, IEEE International Symposium, pp. 68-71, 2007.

[12] CloudSim: A Framework For Modeling And Simulation Of Cloud Computing Infrastructures And Services http://www.buyya.com/gridbus/cloudsim/

[13] E. Cronin, B. Filstrup, A.R. Kurc, and S. Jamin, “An efficient synchronization mechanism for mirrored game architectures," in Proceedings of the 1st workshop on Network and system support for games, pp.67-73, ACM Press, 2002.

[14] E. Cronin, B. Filstrup, and A. Kurc, “A distributed multiplayer game server system," University of Michigan Course Project Report, May 2001.

[15] M. Mauve, S. Fischer, and J. Widmer, “A generic proxy system for networked computer games," in Proceedings of the 1st workshop on Network and system support for games, pp.25-28, ACM Press, 2002.

[16] D. Bauer, S. Rooney, and P. Scotton, “Network infrastructure for massively distributed games," in Proceedings of the 1st workshop on Network and system support for games, pp.36-43, ACM Press, 2002.