2011 IEEE International Conference on High Performance Computing and Communications

A Hybrid P2P System to Support MMORPG Playability

Ignasi Barri, Francesc Giné and Concepció Roig
{ignasibarri,sisco,roig}@diei.udl.cat
Computer Science Department. University of Lleida, Spain

***Abstract*—MMORPG (Massively Multiplayer Online Role Player Games) is the most popular genre among network gamers, and now attract millions of users, who play in an evolving virtual world simultaneously. This huge amount of concurrent players requires the availability of high performance computation servers to provide QoS. Additionally, gaming aware distribution mechanisms are needed to distribute game instances among servers avoiding load unbalances that affect performance negatively. In this work, we tackle the problem of system scalability by means of a hybrid P2P architecture. To distribute game instances of MMORPG over the peers of this architecture, we propose two mapping mechanisms. The *Direct_Mapping* distributes players on demand in order to minimize waiting times. The *Group_Mapping* groups players on queues of different sizes before distributing them, in order to better exploit peers capabilities. The effectiveness and differences in terms of latency, waiting time and reliability of both mapping mechanisms are evaluated through simulation. We show that *Group_Mapping* provides lower latency response, while *Direct_Mapping* distributes game instances faster. In relation to reliability, our results show that the *Direct_Mapping* policy achieves a failure probability under 7% in the worst cases.**

I. Introduction

Massively Multiplayer Online Games (MMOG) are the most popular genre in the computer game world [3]. They can be divided into three categories: *MMORPG* (Massively Multiplayer Online Role Playing Games), *MMORTS* (Massively Multiplayer Online Real Time Strategy) and *MMOFPS* (Massively Multiplayer Online First Person Shooter). Their execution requirements vary with the way of playing in each of them [12]. While MMOFPSs consist of having many isolated game services, with a handful of players each, that are continuously interacting; MMORTSs and MMORPGs consist of a virtual world that never stops and it is always up for any player who wants to satisfy their craving for gaming. Nowadays, of all these genres of on-line games, MMORPGs have become the most popular among network gamers, and now attract millions of users. Thus, the scalability of the computational system to provide MMORPG playability becomes a challenge. This paper is focused on the scalability problem of MMORPG environments taking their characteristics into account.

MMORPGs are characterized by a *Main Game*, which is executed without interruption, and a number of instances, or *Auxiliary Games*, that happen concurrently with the Main Game, that are randomly created on demand by the will of the players. The Main Game is the virtual world, where players can interact between them and the rest of components of the scenario (Non Player Characters and map objects) in order to evolve their characters. The Auxiliary Games[1] are executed out of the Main Game boundaries, involving different ways of playing compared with the Main Game. There are several kinds of Auxiliary Games with different timing, number of players and difficulty requirements. They are created dynamically; so whenever a player wants to play in a specific Auxiliary Game, he has to wait until there are enough players to start it; returning to the Main Game when the Auxiliary Game is over or whenever he/she is fed up with.

Traditionally, computational requirements of MMORPGs have been provided by Client-Server systems. However, when the number of players increases, this approach reaches its limits due to problems of scalability [7]. To overcome the limits of Client-Server systems, the research community has proposed some alternatives with decentralized structures, where each machine contributes to, and benefits from, a large service oriented network. Some works have focused on the Main Game itself to serve MMORPGs. These solutions are based on splitting the Main Game world into different subspaces and distributing them into the decentralized nodes [5] [4], or grouping players according to the load on the map [6]. In this way, each change in the load of the map area, due to the initiation of Auxiliary Games that accumulate large number of players, implies to launch balancing mechanisms to divide and distribute portions of the map to different servers or peers (in distributed proposals). Thus, in dynamic games as MMORPGs are, these proposals are effective but inefficient, due to the fact that they have to balance the load of the map constantly. Unlike these previous works, we propose to distribute Auxiliary Games over a P2P network, while the Main Game is maintained in a central server. This is based on the fact that the load of the Main Game is predictable and stable, whereas Auxiliary Games are more dynamic, less predictable and cause hot spots, which imply peak loads in the overall system.

For these reasons, this paper proposes a hybrid architecture, which is made up of a central pool of servers, where the Main Game is executed, and a set of temporary servers, distributed throughout a P2P topology, where the Auxiliary Games are mapped and executed. For each Auxiliary Game, a new tempo-

This work was supported by the MEyC-Spain under contract TIN2008-05913 and the CUR of DIUE of GENCAT and the European Social Fund.

[1]Dungeons, Raids, and Player vs. Player in the well known MMORPG, World of Warcraft[8].

978-0-7695-4538-7/11 $26.00 © 2011 IEEE
DOI 10.1109/HPCC.2011.80

rary server is chosen among the waiting players for this game. In order to do this, two different issues must be faced up. On the one hand, we have to keep the distributed areas up avoiding disconnections or failures. Thus, a statistical model focused on players' sessions is proposed, in order to assign to each player a faulty likelihood or reliability value, which is used to select those players with lower disconnection probability to act as servers for the Auxiliary Games. On the other hand, the latency response among players of the same Auxiliary Game must be maintained below an acceptable threshold. According to this, the latency among waiting players is calculated, choosing as a temporary server the player with the lowest latency.

To keep the latency values under an acceptable threshold for MMORPG and avoiding server's failures of the Auxiliary Games, this paper proposes two mechanisms to carry out the mapping of Auxiliary Games: *Direct_Mapping* and *Group_Mapping*. They differ from the way of choosing the Auxiliary Game server and, as a consequence, to optimize the latency and reliability capabilities. While Direct_Mapping chooses the Auxiliary Game server among waiting players for such game, the Group_Mapping strategy extends the set of candidates to all the players waiting for such a game and for any kind of Auxiliary Game. Thus, when the set of candidates increases, the likelihood of choosing the best server will be higher.

The effectiveness of these approaches has been evaluated by means of simulation. Our results show that our hybrid architecture is capable of dealing with the scalability problem and maintaining the QoS (latency and fault tolerance) of all the players in system. In addition, we show that *Group_Mapping* provides lower latency response, whereas *Direct_Mapping* distributes game instances faster.

The remainder of this paper is organized as follows. Section II describes the proposed hybrid architecture. Section III exposes the mapping mechanisms to distribute Auxiliary Games. Section IV evaluates the performance of the proposed mapping mechanisms under different lookup criteria. Finally, Section V outlines the main conclusions and future work.

II. Description of the Hybrid Platform

In this section, the proposed system is described globally, discussing its characteristics. Figure 1 shows the architecture of the proposed system that is composed of two main areas: one central area performing the Main Game and a distributed area that grows in a P2P like fashion, where Auxiliary Games are executed.

The components of the central area are the following:

- *Pool of Servers (PS).* They are the main system servers and act as the bootstrap point. A Player number i, P_i, is a client who connects to the PS in order to play a MMORPG.
- *Main Game (MG).* It is the main game, where players interact with the elements of the persistent virtual world. The Main Game is executed in the PS, where the map of the virtual world will be distributed using some specific load balancing mechanism.
- *Auxiliary Games Queue (AGQ).* This is a logical space in the PS used to insert those players, who want to play in any Auxiliary Game (instances). This is a transitory state for players, who will be redirected from the Main Game to the Auxiliary Game. We distinguish five types of AGQ, that ranges from 5 up to 40 players. This range of players has been taken according to the usual values used in real MMORPGs, taking as a representative case the instances of WoW (World of Warcraft) [9].

The distributed area (or Peer-to-Peer area), where the Auxiliary Games are executed, is composed of players machines that are logically grouped in sets, depending on the kind of the Auxiliary Game. So, the distributed area is made up of a set of independent Auxiliary Games scattered throughout the network. Each Auxiliary Game (AG) has the following components:

- *Auxiliary Game Server (AGS).* This is the temporary server of the AG.
- *Auxiliary Game Replicated Server (AGRS).* This is the current replicated server of the AG. This role is used to replace the AGS in case of failure. For this reason, players will play against the AGS and its AGRS, and the AGS will send the game state to both, players and the AGRS. Thereby, the AGRS has the game state constantly updated, ready to replace the AGS if a critical situation requires it. The AGRS assumes that the AGS has crashed when it is not receiving any information from AGS in a very short period of time ($<$ 1 second). So, at this moment, the AGRS becomes the new AGS and the AG is still alive for all distributed players in a transparent way.

III. Mapping Auxiliary Games

The system operation is controlled by the continuous execution of Alg. 1, which has two input flows: new player connections (P_i) and players returning from Auxiliary Games, which have ended. At each iteration, the Pool of Servers checks if there is any player demanding to play to a specific AG (statement P_i demanding any AG_j). If it is the case, the system enqueues the player to the corresponding queue AGQ_j ($PS.toAGQ_j(P_i)$). After this statement, we propose two mapping mechanisms, which try to distribute players of the queue AGQ_j in a different way; they are: *Direct_Mapping* and *Group_Mapping*.

A. Direct Mapping (DM) and Group Mapping (GM)

First of all, if the queue is full of players ($AGQ_j.size() \geq \alpha$) or the queue uptime has been elapsed ($AGQ_j.uptime() \geq \beta$), the system looks for the appropriate servers AGS and AGRS ($AGQ_j.findServer(CRITERIA)$), being *CRITERIA* the target performance capability that is checked: latency or reliability. After that, a new AG is created linking the AGS and AGRS with the rest of players of the queue ($AGS.link(AGQ_j)$ and $AGRS.link(AGQ_j)$).

It is worth pointing out that both mapping policies follow the same procedure explained above. The difference between them lies in the α and β values. A previous work of the authors analyzed the appropriate values for α and β in [1]. While *Direct_Mapping* (DM) has the α value adapted to the

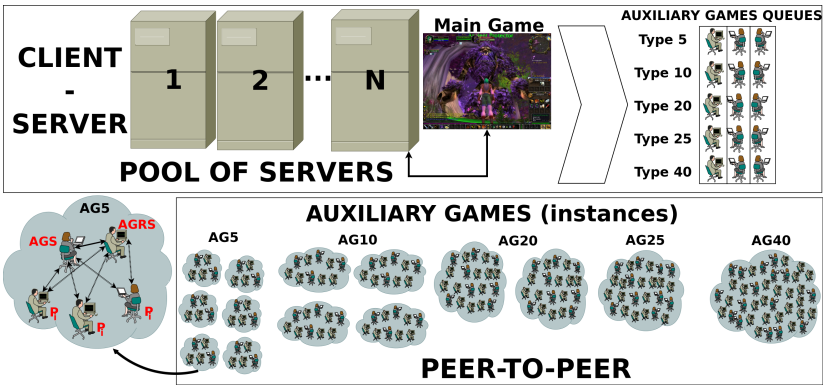

Fig. 1: Hybrid system architecture.

maximum size of the corresponding Auxiliary Game (AG_j), *Group_Mapping* (GM) has a higher α value. Thus, *DM* tries to distribute players, as fast as possible. For this reason, every time that a queue reaches α capacity or β uptime, the waiting players located in this queue are distributed. Meanwhile *GM* groups players in queues with higher α and β values compared with the *DM* parameters, which helps to exploit peers capabilities, in order to improve the latency or reliability performance. The discussion of the appropriate α and β values is conducted in Section IV.

Input: $\forall P_i$ connecting to PS
Input: $\forall P_i$ returning from any AG_j to PS

foreach P_i *demanding* AG_j **do** $PS.toAGQ_j(P_i)$;
if $((AGQ_j.size() \geq \alpha)$ *and* $(AGQ_j.uptime() \geq \beta))$ **then**
    AGS = $AGQ_j.findServer(CRITERIA)$;
    AGRS = $AGQ_j.findServer(CRITERIA)$;
    $AGS.link(AGQ_j)$;
    $AGRS.link(AGQ_j)$;

Algorithm 1: *Mapping Main Algorithm for Direct and Group Mapping.*

In order to maximize the performance in terms of latency and reliability for both mapping mechanisms, *DM* and *GM*, we propose the two following options to be used as *CRITERIA* for finding a new server (AGS) and replicated server (AGRS):

- *Latency Lookup (LL):* It checks the peer-to-peer latency of all players located in a specific queue AGQ_j. Then, it is selected, as the *AGS*, the player who has the lowest latency with respect to the rest of players of the AGQ_j, being the *AGRS* the player with the second lowest latency value.
- *Probability of Disconnection (PD):* According with the estimation of players uptime, the faulty likelihood is calculated. Then, it is checked player by player their worst predicted uptime values, chosen as AGS and AGRS, those players with the highest predicted uptime, i.e., the most reliable players.

B. Players Behaviour Model

In order to find the best reliable player to act as a server, the understanding of the behaviour of MMORPG players and their subsequent modelling is a key issue of our work. This section discusses the model in terms of players uptime behaviour.

The modelling of players uptime is based on their history. Thus we are able to predict how much reliable will be a player in order to select the adequate AGS and AGRS in terms of fault tolerance. From [9], we have obtained a mean and standard deviation values of 2.8 and 1.8 respectively. These values can be modelled by a gamma distribution with shape (K) equal to 1.157143 and a scale (Θ) equal to 2.419754, given that its has been successfully evaluated with the *Chi-square test*, indicating the goodness of our model, in statistics terms. Then, with this model, we are able to assign a faulty likelihood to any player taking as a reference the player uptime.

This model is used in order to select the server (AGS) and its replicated (AGRS) for each Auxiliary Game (AG) adequately. It is a fault tolerance based model, which implies that despite the existence of extreme players behaviour, in terms of sudden disconnections or large uptimes, we prioritize to select those players with better and more stable connection settings, in order to ensure a good reliability in the distributed area.

IV. Results and Discussion

In this Section, experimentation is conducted to demonstrate the feasibility and good performance of the proposed mapping mechanisms and to show a comparison between them. The experimentation was performed through simulation using SimPy [10]. SimPy is a discrete-event simulation language based on standard Python. SimPy tools have been used to implement nodes of the platform, which can fulfil four distinct roles: player, AGS, AGRS and PS. The SimPy procedures allow to simulate randomness of the real behaviour of players.

Each simulation consisted of $50,000$ players connecting sequentially to PS in a balanced Main Game. Every time that a player was created, this selected the kind of Auxiliary Game (AG_j), which wants to play. It is worth pointing out that an AG_j means that j concurrent players are playing the same Auxiliary Game without any interaction with the rest of players of the Main Game. Taking the offer of the existing

	AG_5	AG_{10}	AG_{20}	AG_{25}	AG_{40}
Percentages (%)	66	14	2	3	15

TABLE I: Percentages of AG_j modelled into the simulation.

Parameters	AG_5		AG_{10}	
	DM	**GM**	**DM**	**GM**
α (No.)	5	50	10	300
β (sec.)	-	200	-	720

TABLE II: α and β values for *DM* and *GM*.

World of Warcraft' Instances [11] into account, five different AGs were modelled: AG_5, AG_{10}, AG_{20}, AG_{25} and AG_{40}, together with its associated queues AGQ_j. According to the statistics of the World of Warcraft game [11], Table I shows the percentage of each kind of AG_j in relation to the total of AGs. Taking these percentages into account, this experimentation is focused in the results related to AG_5 and AG_{10} games, given that they represent the 80% on total AGs and as a consequence, they have the highest impact onto the global performance of the game.

In order to verify that the distributed area is dynamically adapted to the on-demand queries of players, players' connections were sequential with constant inter-arrival time ($\approx$ 1 second) to submit the mapping mechanisms to a constant stress situation or constant peak load. Another important issue is the calculus of the players' latency against the PS. This was determined by a triangulated heuristic, delimiting the 2-Dimensional Euclidean Space to ($x = [-1000, +1000], y = [-1000, 1000]$). This methodology is based on the relative coordinates explained in [2]. Furthermore, the players' uptime model is also explained in Section III-B.

Table II shows the chosen α and β values for both proposed policies (*DM* and *GM*) and for the queue of each type of game (AG_5 and AG_{10}). For the *DM* policy, the α value of each type of AG has been established equal to its maximum capacity, whereas β value is irrelevant; it means that it is the elapsed time to reach the α players in each queue. In relation to *GM* policy, the α values has been increased to 50 and 300 players, this new α value improves the system performance avoiding excessive waiting time for players; meanwhile, the waiting time (β) was extended for AG_5 and AG_{10} until 200 and 720 seconds, respectively. These values were chosen taken into account that a player who wants to play in an AG_5 is prepared to wait less than a player who wants to play in AG_{10}.

Finally, we modelled the players uptime, taking the gamma distribution into account, which models the different players' uptime, divided with an hour intervals. Figure 2 contains a table, which shows numerically the histogram of the data, also plotted in the same Figure. In order to introduce randomness in the mentioned intervals, we chose a normal distribution with mean equal to the time of the gamma distribution probability, and the variance equal the square of the standard deviation used previously. This operation determines the variability of players connections.

Hours	%
[0,1]	12.808
]1,2]	26.384
]2,3]	23.456
]3,4]	16.618
]4,5]	9.776
]5,6]	5.318
]6,7]	2.91
]7,8]	1.158
]8,9]	0.786
]9,10]	0.362
]10,11]	≈ 0

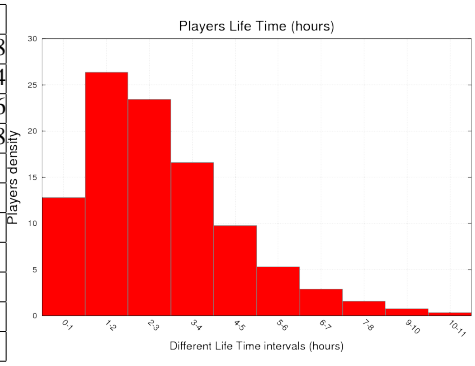

Fig. 2: Gamma Distribution Histogram, and summary table by an hour interval.

Scalability and QoS Evaluation

The scalability of the proposal indicates its ability to create AGs on demand, while the QoS of the whole system, measured in terms of latency and reliability, is maintained.

Table III shows the average (AVG) and standard deviation (SD) for *DM* and *GM* mechanisms, when they apply the latency (*LL*) as a criteria of distribution. This table shows, for the mentioned mapping mechanisms and for the AG_5 and AG_{10} types, their performance in terms of the number of created AGs, the players' latency and the waiting queue time. In each case, we evaluated 1000 different simulations.

As it can be observed, the number of created AGs is strongly correlated with the AGs percentages commented at the beginning of Section IV. This means that the mapping mechanisms are able to distribute to the P2P area the sudden concentration of players caused by AGs. Thus, the PS load remains stable, with no need to provide extra computational resources to manage these instances. Regarding latency, the best performance is achieved by the *GM LL* mechanism, which is able to subtly reduce the latency $\approx 1.5\%$ and $\approx 2.73\%$ in relation to *DM LL*. However, this slight improvement in latency terms has significant negative effects on the waiting queue times. It is shown that the *GM LL* mapping mechanism increases the waiting times $\approx 1920\%$ and $\approx 886\%$. Thus, the simulation results let us to not consider *GM* as an valid option to use in a real environment. For this reason, we are going to focus our subsequent analysis on the *DM* mechanism.

Next, we have evaluated the latency of *DM LL* policy in relation to the traditional Client-Server architecture, which means that players play against the pool of servers directly instead of playing against the AGS as we propose. Likewise, we have proposed a modification of the *DM LL* policy, called *DM LL modif*, which consists on selecting, as AGS of a specific AG_j, the player, waiting into the associated AGQ_j, with the lowest latency in relation to the central Pool of Servers. Note that the application of *DM LL modif* only implies j comparisons in front of the $j \cdot \frac{(j-1)}{2}$ comparisons of our original *DM LL*, being j the length of the AGQ_j. The comparison of the obtained players' latency under the three methods is shown in Figure 3. From this Figure, we can see as the *DM LL* obtains an improvement, in relation

Mapping	Criteria LL		Number AGs		Latency (ms)		Queue Time (s)	
			AG_5	AG_{10}	AG_5	AG_{10}	AG_5	AG_{10}
DM	LL	AVG	6609	699	828.38	805.25	4.024	33.201
		SD	23.061	9.321	3.132	5.620	0.01	0.477
GM	LL	AVG	6590	702	814.634	785.39	77.277	294.38
		SD	18.97	6.89	1.476	3.941	0.193	2.909

TABLE III: Mapping policies comparison.

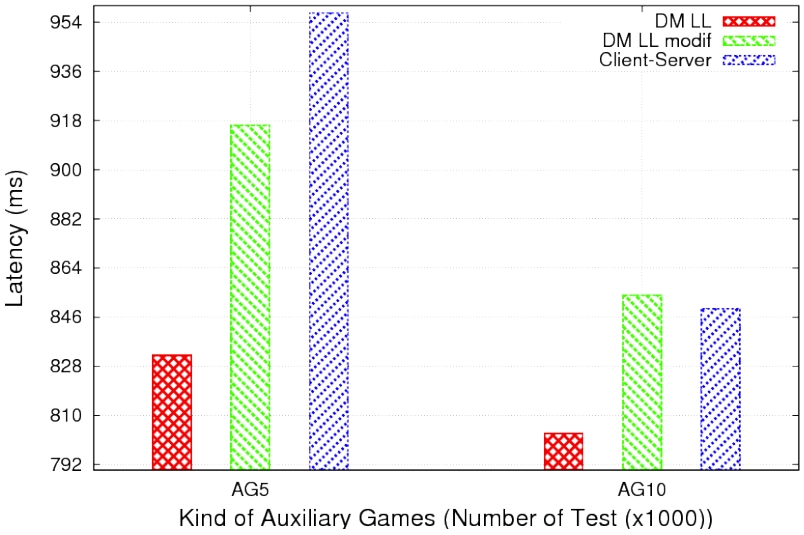

Fig. 3: Latency comparison between *DM LL*, *DM LL modif* and *Client-Server*

to the Client-Server architecture, of 15% and 7% for AG_5 and AG_{10}, respectively. Thus, our hybrid model improves the classical Client-Server model in latency terms. In relation to the modified *DM LL*, we can see as *DM LL* improves an 10% and 6% the *DM LL modif* for AG_5 and AG_{10}, respectively. From these trends, we can see as the gain decreases with the queues' length (j), whereas the computational cost increases in the same proportion. Therefore, the interaction between the computational cost curve with the players' latency reveals that the application of this modification has sense whenever Auxiliary Games have a high number of players ($>$ 30 players).

Regarding the appropriateness of the *PD* criteria to look for AGS and AGRS servers for AG_j, we studied the reliability for *DM PD* mechanism. The *PD* criteria looks for the best reliable player located in the queue AGQ_i. It checks the lowest predicted uptime of each waiting player and it selects the two players with highest uptime for acting as AGS and AGRS, respectively. Figure 4 shows graphically with dashed lines, the lowest players' uptime and the chosen AGS and AGRS, among the different players for AG_5 and AG_{10}.

Table IV shows the AVG and SD of the faulty likelihood of the AGS and AGRS. In addition, we show the faulty likelihood of the AG, which is calculated as the product of the likelihood of AGS and AGRS. As it can be observed, in both cases, the faulty likelihood of AG is under 7%. Note that this performance is achieved by the inclusion in each AG of an Replicated Server (AGRS), which enhances the fault tolerance mapping mechanisms. It is worth remarking that these percentages show the highest faulty likelihood. So, it shows the worst performance case, which reveals the goodness of the *DM PD* mechanism combined with the role of the AGRS.

V. Conclusion and Future Work

This paper proposes a hybrid architecture, made up of a central and a distributed area, to tackle the scalability challenge of MMORPG games. An MMORPG is characterized by a Main Game, which is executed without interruption, and a number of Auxiliary Games, which are randomly created by the will of players. According to this, this paper proposes an hybrid architecture to play MMORPGs in such a way that the Main Game is executed in the central area, whereas the Auxiliary Games are mapped into the P2P area.

Additionally, we defined two mapping mechanisms to distribute Auxiliary Games into the P2P area: the *Direct_Mapping* mechanism that distributes players on demand and the *Group_Mapping* mechanism to distribute players from the most populated waiting queues; in order to better exploit peers capabilities. To maximize the performance in terms of latency and reliability for both mapping mechanisms, we proposed latency or reliability options to be used as *CRITERIA* for finding a new server and replicated server for the *Auxiliary Games* of the distributed area. By means of simulation, it has been demonstrated that the proposed mapping mechanisms are able both to provide a distributed

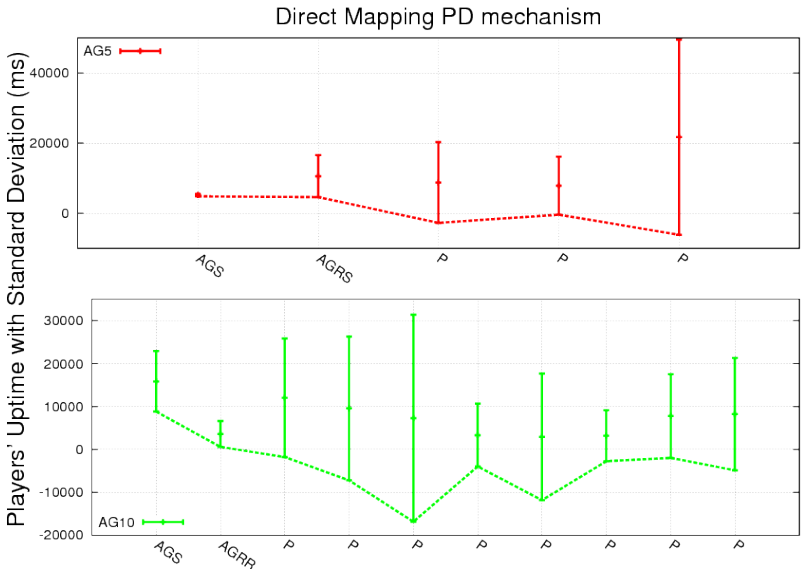

Fig. 4: AGRS AGS' selection process for AG_5 and AG_{10} auxiliary games.

DM PD performance		AG_5			AG_{10}		
		AGS	**AGRS**	**AG**	**AGS**	**AGRS**	**AG**
Faulty likelihood (%)	AVG	32	12	4	28	24	7
	SD	10	14	-	8	11	-

TABLE IV: *DM PD* reliability performance for AG 5 and AG 10.

platform that scales on demand keeping the latency values under an acceptable threshold. Regarding differences, we show that *Group Mapping Latency Lookup* optimizes subtly the latency, meanwhile the *Direct Mapping Latency Lookup* is able to distribute players faster and with enough QoS. Likewise, we have shown as our proposal is able to achieve lower average latencies compared with the traditional Client-Server architecture. About reliability, it has been demonstrated, that the *Direct Mapping* is able to achieve a failure probability under 7% in the worst cases.

Future work is oriented to face the cheating problem related to MMORPG games in order to ensure a good player experience in the overall game. In order to better exploit the peer's resources, another improvement avenue will be to manage the inherent heterogeneity of players. Finally, another interesting issue will be to merge our current mapping mechanisms with market criteria to reward the AGS and AGRS, given that, they are sharing their resources in an altruistic way.

REFERENCES

[1] I. Barri, F. Giné, and C. Roig. A Scalable Hybrid P2P System for MMOFPS. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference*, 2010.

[2] T. S. Eugene and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM*, 2001.

[3] Research in China. China Online Games Market Report, 2008. http://www.researchinchina.com/Htmls/Report/2008/1944.html.

[4] J. Keller and G. Simon. Solipsis: A Massively Multi-Participant Virtual World. In *PDPTA*, 2003.

[5] B. Knutsson, Honghui Lu, Wei Xu, and B. Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. In *INFOCOM*, 2004.

[6] Hu. Liu and Y. Lo. DaCAP - A Distributed Anti-Cheating P2P Architecture for Massive Multiplayer On-line Role Playing Game. In *CCGRID*, 2008.

[7] V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D. Epema, and T. Fahringer. Efficient management of data center resources for massively multiplayer online games. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 2008.

[8] M. Suznjevic, O. Dobrijevic, and M. Matijasevic. MMORPG Player Actions: Network Performance, Session Patterns and Latency Requirements Analysis. *Multimedia Tools Appl.*, 2009.

[9] P. Tarng, K. Chen, and P. Huang. An Analysis of WoW Players Game Hours. In *NetGames*, 2008.

[10] IBM Developers Works. Charming Python: SimPy Simplifies Complex Models (Simulate Discrete Simultaneous Events for Fun and Profit). 2002.

[11] WoWWiki. Your Guide to the World of Warcraft, 2011. http://www.wowwiki.com/Instances_by_level.

[12] M. Ye and L. Cheng. System-Performance Modeling for Massively Multiplayer Online Role-Playing Games. *IBM Syst. J.*, 2006.