Multimed Tools Appl (2016) 75:2005–2029
DOI 10.1007/s11042-014-2389-0

Distributing game instances in a hybrid client-server/P2P system to support MMORPG playability

Ignasi Barri · Concepció Roig · Francesc Giné

Received: 23 October 2013 / Revised: 21 October 2014 / Accepted: 14 November 2014 /
Published online: 28 December 2014

Abstract MMORPG (Massively Multiplayer Online Role Playing Games) is the most popular genre among network gamers, and now attract millions of users, who play simultaneously in an evolving virtual world. This huge number of concurrent players requires the availability of high performance computation servers. Additionally, gaming aware distribution mechanisms are needed to distribute game instances among servers to avoid load imbalances that affect performance negatively. In this work, we tackle the problem of game distribution and scalability by means of a hybrid Client-Server/P2P architecture that can scale dynamically according to the demand. To manage peak loads that occur during the game, we distribute game computation across the system according to the behavior of MMORPGs. We distinguish between the computation associated with the Main Game, that affects all players, and the computation of Auxiliary Games that affects only a few players and acts in isolation from the execution of the Main Game. Taking this distinction into account, we propose a mechanism that is focused in the distribution of Auxiliary Games, as an entity, across the pool of servers and peers of the underlying hybrid architecture. We evaluate the performance of the balancing mechanism taking the criteria of latency and reliability into account, and we compare the effectiveness of the mechanism with a classic approach that applies load balancing to individually players in a Client-Server system. We show that the balancing mechanism based on the *latency criteria* provides lower latency

I. Barri · C. Roig · F. Giné (✉)
Computer Science Department, University of Lleida, Lleida, Spain
e-mail: ignasibarri@diei.udl.cat

C. Roig
e-mail: roig@diei.udl.cat

F. Giné
e-mail: sisco@diei.udl.cat

than the classical proposal, while in relation to reliability, we obtain a failure probability of under 0.9 % in the worst case, which is amply compensated by the scalability provided by the use of the P2P area.

Keywords MMORPG · Hybrid system · P2P · Load balancing

1 Introduction

Massively Multiplayer Online Games (MMOG) are the most popular genre in online computer games [29]. 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 them [37]. While MMOFPSs consist of many isolated game services with a handful of players each, who are continuously interacting, MMORTSs and MMORPGs consist of a virtual world that never stops and is always up for any player who wants to satisfy their desire 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 QoS and scalability of the computational system to ensure MMORPG playability is a challenge. This paper focuses on these problems in 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, randomly created on demand by the players. The Main Game is the virtual world, where players can interact each other and the other components of the scenario (Non-Player Characters and map objects) in order to evolve their characters. The Auxiliary Games[1] are executed outside the Main Game boundaries and involve 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 players want to play in a specific Auxiliary Game, they have to wait until there are enough players to start it; returning to the Main Game when the Auxiliary Game is over or whenever they tire of it.

The most common way to provide service to MMORPGs is based on a centralized management provided by Client-Server structures, usually based on cluster platforms [10]. To overcome the scalability limits of these Client-Server systems, when the number of players increases dramatically, we propose in this paper the use of a hybrid environment that is composed of a centralized cluster system and a distributed P2P area. In addition, we develop a mapping mechanism to achieve load balancing and scalability of the MMORPGs in this hybrid platform, in which the Main Game is maintained in the central cluster of servers and Auxiliary Games are used as the indivisible entities, where the mapping is applied. This is based on the fact that the load in the Main Game has few fluctuations and so, it can be predicted, whereas Auxiliary Games are more dynamic, less predictable and also cause hot spots, which imply peak loads in the overall system. Additionally, players involved in an Auxiliary Game are continuously interacting. Thus, the movement of all players in an Auxiliary Game, as an indivisible entity, to the same server avoids

[1] Dungeons, Raids, and Player vs. Player in the well known MMORPG, World of Warcraft [33, 36].

communications between nodes. Then the mapping mechanism acts at two levels. Firstly, it balances Auxiliary Games among the servers in the central cluster in order to manage computation imbalances. Secondly, when the whole central cluster is overloaded, the mapping assigns Auxiliary Games to the P2P Area. Thus, for each Auxiliary Game to be distributed to the P2P Area, a new temporary server is chosen among the players waiting for this game.

In order to choose the temporary server, two different issues must be faced. On one hand, we have to keep the Distributed Area up to avoid disconnections or failures. Thus, a statistical model focused on the players' sessions history is proposed in order to assign each player a fault likelihood or reliability value, which is used to select those players with lower disconnection probability. On the other hand, the latency response among players in the same Auxiliary Game must be maintained below an acceptable threshold. According to this, the latency among waiting players is calculated, with the player with the lowest latency being chosen as a temporary server for the Auxiliary Game.

The effectiveness of the proposed distribution mechanism over the hybrid architecture is evaluated by simulation. The effects of balancing Auxiliary Games, instead of players, among servers, is evaluated by comparing our mechanism with a representative case of the classic load balancing approach presented by Bezerra et al in [6]. Additionally, we show that our system is able to scale properly when the number of Auxiliary Games in the system increases due to the rise in players. This scalability is achieved by maintaining the QoS in terms of latency and fault tolerance.

The remainder of this paper is organized as follows. Section 2 reports the main contributions of the literature about distribution mechanisms of MMORPGs. Section 3 describes the hybrid system with the balancing mechanism. Section 4 analyses the viability of the P2P area for computing Auxiliary Games. Section 5 evaluates the performance of the global system. Finally, Section 6 outlines the main conclusions and future work.

2 Related work

The techniques that are reported in the literature to give service to MMORPGs vary according to the kind of system, centralized or distributed, used for executing the game. Centralized architectures are traditionally based on cluster systems. The distribution techniques for game computation in such kinds of system are mainly based on splitting the game world map into different subspaces, or cells, and distributing these among the nodes in the cluster [6, 23, 24, 26]. After the initial assignment, these cells will be dynamically reassigned among servers during the game, to respond to changes in the load caused by players' movements into the game world [4, 11]. The frequency of these rebalancing operations is heavily affected by the size of the cells. Big cells can imply significant differences in the number of players assigned to each one, and so, in order to achieve load balancing, the initial distribution of cells among servers is more complex. However, players change cell less often, and this avoids some dynamic reassignments during the game. The option of small size cells (microcells) facilitates the initial load balancing distribution but causes greater number of cell movements by players during the game, which implies more dynamic redistribution operations and thus more overhead in the run time [12, 17].

An interesting work in the line of balancing microcells of players between servers is the paper of Bezerra et al. in [6]. Bezerra proposes a balancing schema which considers the upload bandwidth of the server as the load to distribute, what is done between

servers with the aim of reducing the inter-server communication overhead by using a greedy graph partition growing algorithm.The mechanism starts when a server in the cluster of servers is overloaded. This server selects a number of other servers to become involved with the distribution. First, it chooses the least loaded server among its neighbors and sends a request for it to participate in the load balancing. The chosen server rejects the request if it is already involved in another balancing group, otherwise it responds with the load information of its own neighbors. If the selected neighbor server is unable to absorb all the extra workload of the initiating server, the selection is performed again among the neighbors not only of the overloaded server, but also the neighbors of the already selected servers. The selection continues until the first server's workload can be absorbed. This work will be used through this paper as a comparison case given that it is a representative approach of player-based load balancing mechanisms in Client-Server systems applied to MMORPGs.

An additional aspect that is considered by some authors in the splitting process is the Area of Interest (AOI) of players [3]. The AOI is the physical area of the world map whose information about game state is relevant for a specific player. Thus, the assignment of players of the same AOI into the same cell will diminish the number of communications across the network.

In the case of MMORPGs, the focus of this paper, we propose splitting the world map based on the Auxiliary Games, instead of cells of a fixed size, for the two following reasons: (a) An Auxiliary Game constitutes a clear AOI where players communicate with each other and, (b) the computation associated with the game for the limited number of players that usually constitutes an Auxiliary Game can be executed in a single current server.

The game distribution techniques, mentioned above, usually reported for centralized systems, present a common problem of scalability when they are serving MMORPGs. This is due to the unpredictable behavior of players, which sometimes creates peak load situations that cannot be solved by system servers. With the goal of providing an unlimited scalability that is able to manage peak load situations dynamically, some authors propose the use of completely decentralized systems, such as Peer-to-Peer networks. In this kind of system, apart from the load distribution techniques, there are additional aspects to be faced [15] that have been studied by several authors: (a) Establishing an effective mechanism for propagating events in a high latency network. Different alternatives have been presented that are applied in the network layer [5, 8] or the application layer [9]. (b) Ensuring data persistence in the game world map [19]. (c) Management of the cheating problem among the peers [27, 31]. (d) Applying incentive mechanisms that promote the participation of peers and avoid freeriders [21, 34].

The load distribution techniques developed for MMORPGs in Peer-to-Peer systems are mainly focused on the decentralized management of the AOI of the players. Some authors have developed techniques for distributing the game load based on AOI [7, 13, 28]. However, in an MMORPG, apart from updating their AOI, players also frequently need to update their view of the virtual game world, causing a high communication overhead for such players. To alleviate this overhead in completely decentralized systems, some authors propose the inclusion of some additional high capacity server to manage the global game world, where most of the players are located [30, 32, 38].

Following this line, we propose a hybrid architecture that combines a centralized and a Peer-to-Peer system, to give service to MMORPGs. This allows the execution of tasks related to the global game world in the centralized server and other tasks corresponding to the Auxiliary Games in the Distributed Area of the system to be combined. In the following

section, we present our proposed hybrid system and the load distribution techniques for MMORPGs.

3 The client-server/P2P hybrid system

This section describes the proposed hybrid system, discussing its characteristics and details of its mechanisms. The architecture of the system is explained in Section 3.1, while the distribution policy of Auxiliary Games over the hybrid architecture and mechanisms of establishment and maintenance of the Distributed Area are described in Section 3.2. Finally, Section 3.3 analyzes the communication cost caused by the application of our balancing algorithm.

3.1 The hybrid architecture

Figure 1 shows the architecture of the proposed system, which is composed of two areas: (a) one *Central Area* performing the Main Game and the Auxiliary Games and, (b) a *Distributed Area* that grows in a P2P like fashion, where those Auxiliary Games that cannot be served by the Central Area for overload reasons are executed.

The components of the Central Area are the following:

- *Cluster of Servers (CS).* This is composed of a set $\{S_i \mid 1 \leq i \leq N\}$ of N Servers, which are the main servers in the system and act as the bootstrap point. Thus, each player, P_j, requesting to enter the system, will attempt to connect to it. A Server S_i is a physical computer that manages a part of the Main Game and some Auxiliary Games, providing service to a limited number of players according to its capabilities. The maximum capacity of a server is determined by the number of concurrent players which is able to serve, named max_S_load, which is assumed to be the same for each server S_i. Likewise, we consider the CS overloaded when at least one Server S_i of the CS is overloaded.
- *Main Game (MG).* This is the principal game, where players interact with the elements of the persistent virtual world. The Main Game is executed in the CS, where the map

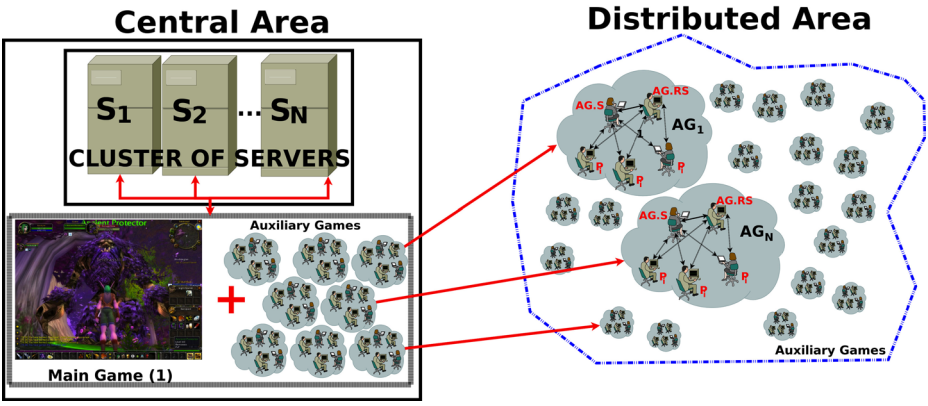

Fig. 1 Hybrid system architecture

of the virtual world is distributed using a specific load balancing mechanism described and evaluated in the following sections.

– *Auxiliary Games (AG_x).* These are the game instances that are executed independently from the Main Game. There are different kinds of AG_x in function of the number of players in each, which is denoted by $AG_x.size$. Therefore, each AG_x is defined as a set $\{P_j \mid 1 < j \leq AG_x.size\}$ of players facing a specific mission. Our proposal in overload cases is focused on the efficient distribution of AG_x over the non-overloaded servers, considering them as indivisible entities for allocation.

The Distributed Area (or Peer-to-Peer area) is the area, where the extra-number of players belonging to the Auxiliary Games that cannot be served in the Central Area, are mapped. It is composed of players' machines that are logically grouped into different kinds of AG_x, each of a specific size and isolated from the rest of the AG_x. Thus, the Distributed Area is made up of a set of Auxiliary Games that conform an independent P2P subarea, each scattered across the network and without any kind of communication between each P2P subarea. To manage the execution of each AG_x, the corresponding P2P subarea will have the two following servers:

– *Auxiliary Game Server ($AG_x.S$).* This is the temporary server of the AG_x. It is worth pointing out that any player of the AG_x can be chosen as a temporary server. The experimentation carried out in Section 4 shows that a normal personal computer is able to serve, at least, up to 40 players without problems of computational power, memory or bandwidth usage. Note that 40 players is the maximum Auxiliary Game size accepted by the majority of MMORPGs [33]. In this way, players' machines can be used to serve a single Auxiliary Game with a satisfactory *QoS*.
– *Auxiliary Game Replicated Server ($AG_x.RS$).* This is the current replicated server of the AG_x. This role is used to replace the $AG_x.S$ in case of failure. For this reason, players in the AG_x will play against the $AG_x.S$ and its $AG_x.RS$, and the $AG_x.S$ will send the game state to both, players and the $AG_x.RS$. Thus, the $AG_x.RS$ has the game state constantly updated, ready to replace the $AG_x.S$ if a critical situation requires it.

According to the system described, the next section introduces a new load balancing mechanism for distributing Auxiliary Games, as an indivisible entity, over the Central and Distributed Area to provide scalability to the system according to the demand.

3.2 Load balancing over the hybrid architecture

The balancing approach presented in this paper is established under three premises: (a) servers of the CS are considered homogeneous, (b) the load of each server S_i is proportional to the number of players connected to it and, (c) the P2P Area is exploited when the CS is unable to deal with an overload situation.

The load balancing methodology acts whenever a server from the CS reaches its maximum load capacity (max_S_load). In this case, the balancing mechanism is able to decide if the necessary number of players that causes the overload situation ($extra\_S\_players$) has to be: (a) balanced to another available server in the CS or, (b) distributed to the P2P Area. The methodology of the load balancing proposal is shown in Algorithm 1. Note that in our load balancing mechanism, this distribution is applied to an $extra\_S\_players$ players with an extra plus of 10 % of the maximum server capacity in order to minimize the number of distributions. Therefore, the total number of players to be balanced or distributed is $(extra\_S\_players + 10\,\% \cdot max\_S\_load)$.

```
Input: extra_S_players = extra_S_players + 10% · max_S_load;
Output: CS Area, P2P Area;
begin
    while (CS is overload) do
        foreach ({S_i} ∈ CS) do
            if (S_i.load ≥ max_S_load) then   /* Balancing to Central Area */
                foreach ({S_j} ∈ CS − {S_i}) do
                    while (S_j.available_load ≥ AG_x.size) do
                        S_i.balances(S_j, AG_x);
                        extra_S_players − = AG_x.size;
                    end
                end
            end
            while (extra_S_players > 0) do    /* Distributing to P2P Area */
                S_i.distributes(P2P, AG_x);
                extra_S_players − = AG_x.size;
            end
        end
    end
end
```

Algorithm 1: Distribution mechanism of Auxiliary Games in the hybrid system.

Algorithm 1 is executed while the CS is overloaded. This means that at least one server in the CS is overloaded. Therefore, each server in the CS is checked for overloading. After this statement, the load balancing is able to: (a) balance the extra-load to another available server in the CS of the Central Area or, (b) distribute it to the P2P Area.

```
S_i.distributes(P2P, AG_x):
Input: P2P Area, AG_x;
Output: AG_x.S, AG_x.RS;
begin
    foreach ({P_i} ∈ AG_x) do
        AG_x.S = AG_x.findServer(CRITERIA);
        AG_x.RS = AG_x.findServer(CRITERIA);
    end
    link(AG_x,AG_x.S,AG_x.RS);
end
```

Algorithm 2: Function to look for a Server and Replicated Server between players of the Auxiliary Game.

For the Central Area, the algorithm checks if there is any server $S_j \in CS$ able to accept players belonging to the $extra\_S\_players$. If so, a complete AG_x of the overloaded server S_i is balanced to this new server S_j ($S_i.balances(S_j, AG_x)$). Note that our proposal always distributes complete Auxiliary Games, so the inter-server communication is diminished significantly. Next, if there is no other Server $S_j \in CS$ able to take in more extra players, ($extra\_S\_players > 0$), then these remaining players will be distributed to the P2P Area by means of the *distributes* function. Note that the cost of Algorithm 1 is $\theta(N^2)$, where N is the number of servers of the Central Area. This cost can be considered negligible taking two different aspects into account: (a) the value of N of a typical MMORPG CS is usually limited below 10^3 [25] and (b) this algorithm is only launched punctually, whenever the CS is overloaded.

In Algorithm 2, the *distributes* function looks for a Server $AG_x.S$ and a Replicated Server, $AG_x.RS$, among the players in the Auxiliary Game. This algorithm is able to vary the *CRITERIA* used to search for the optimal $AG_x.S$ and $AG_x.RS$ depending on the parameter of QoS to be optimized. The two following options were considered for use as *CRITERIA*:

- *Latency Lookup (LL):* This checks the network latency of each player in relation to the rest of players in the same AG_x. Then, it selects as the as the $AG_x.S$, the player who has the lowest latency with respect the others. The $AG_x.RS$ will be the player with the second lowest latency value. The additional cost of our policy is the number of comparisons to be carried out to find the servers with the minimum latency, which is the $(AG_x.size) \cdot (AG_x.size - 1)/2$; in the worst case being an $AG_x.size = 40$, it is equal to 780 comparisons, which, taking the power of a current CPU into account, can be considered negligible.
- *Probability of Disconnection (PD):* According with the estimation of players' uptime, the fault likelihood of each player is calculated. Then, this is checked player by player, choosing those players with the highest predicted uptime, i.e., the most reliable players, as the $AG_x.S$ and $AG_x.RS$ respectively. In order to do so, for each player $P_j \in AG_x$, the minimum and maximum predicted uptime ($P_j.Uptime$) is calculated by applying the mean and standard deviation based on his historical behavior, giving the following interval:

$$\left[P_j.Uptime_{min}, P_j.Uptime_{max}\right] \tag{1}$$

With these values of players' uptime, the PD criteria selects, for acting as the $AG_x.S$, the player with the highest minimum uptime, such that:

$$AG_x.S.Uptime_{min} = max\left\{P_j.Uptime_{min} | \{P_j\} \in AG_x\right\} \tag{2}$$

The replicated server $AG_x.RS$ is selected with the same criteria, excluding the player that acts as $AG_x.S$ from the selection set. Then:

$$AG_x.RS.Uptime_{min} = max\left\{P_j.Uptime_{min} | \{P_j\} \in AG_x - \{AG_x.S\}\right\} \tag{3}$$

Thus, we are working with a pessimistic assumption. Likewise, it is worth remarking that this method takes the average plus its deviation into account, which is more reliable, in statistical terms, than taking only the average into account. In order to minimize the storage information of the historical of each player and taking previous studios [20] into account, we need to maintain a maximum of 50 records per each player to obtain an estimate accurate enough.

Section 5.2.1 analyzes the QoS of the players according to the chosen latency and reliability *CRITERIA*.

Once the $AG_x.S$ and $AG_x.RS$ have been found, the *link* function of Algorithm 2 creates the P2P subarea of AG_x with an interconnection schema of players' machines, as this of Fig. 2, which corresponds to a bipartite graph $G = \{S \cup P, E\}$, where:

- S is the set of two players machines acting as servers. $S = \{AG_x.S, AG_x.RS\}$.
- P is the set of remaining players machines in AG_x. $P = AG_x - S$.
- E is the set of all possible edges (S_i, Pj) such that $S_i \in S$ and $P_j \in P$, plus the edge $(AG_x.S, AG_x.RS)$.

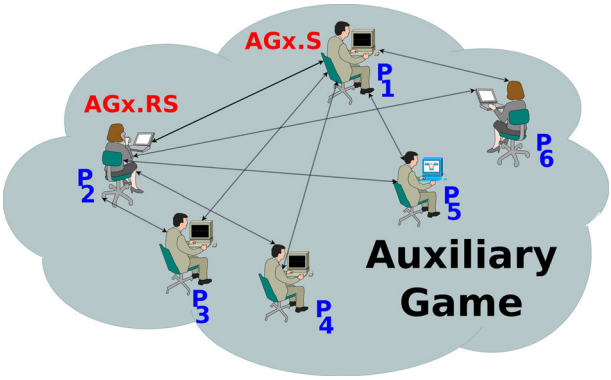

Fig. 2 An enlarged view of an Auxiliary Game (AG_x) structure in the Distributed Area

The two servers of S will maintain the list of players, making up its Auxiliary Game AG_x, together with the main players' characteristics (latency and reliability). Once the Auxiliary Game is moved to the P2P area, it continues its execution until it finishes. Then, their players can return to the CS to continue with the game.

After the application of this distribution mechanism, the overloaded servers of the Central Area reduce their workload. This allows the system to continue accepting new players above its CS capacity, providing a system that is able to grow according to the demand, exploiting the peers' capabilities by means of distributing them to a self-organized P2P game area.

3.2.1 Management of the distributed area

The management for the Distributed Area is based on the standard functionalities applicable to any P2P system, focused on the specific case of using the defined P2P subareas for running Auxiliary Games. So, the mechanisms of insertion of peers and resource discovering are not needed in this context, because the players conforming the AG_x are known beforehand, and no new players will be added during the execution of the Auxiliary Game. Thus, the management policies to be applied are restricted to the maintenance of each P2P subarea and the exit of peers. These are described next.

The maintenance of each AG_x subarea is performed by Algorithm 3 in each period of time T. Control message exchanging is performed between both servers in S to the rest of players of the Auxiliary Game AG_x belonging to the set P. In this way, $AG_x.S$ and $AG_x.RS$ check the state of the total underlying players managed by the $AG_x.S$. Likewise, the Auxiliary Game Server $AG_x.S$ notifies the state information of the AG_x to the Cluster of Servers (CS) of the Central Area, keeping the global state system updated. In addition, each player P_j replies to its Servers by sending information about its state. This happens whenever the player P_j had not sent another message to $AG_x.S$ and $AG_x.RS$ during the same interval T previously. The cost of the algorithm is $\theta(AG_x.size)$, where $AG_x.size$ is the number of peers in the Auxiliary Game. It is worth pointing out that the exact value of T depends of the nature of the MMORPG, although it should be lower than $1s$ in order to replace the Server of the Auxiliary Game, if it was necessary, in an inadvertently way for players [33].

```
Input: AG_x;
begin
    foreach (T ms) do
        foreach ({P_j} ∈ AG_x) do
            if (P_j==AG_x.S) then
                AG_x.S sends(P ∪ {AG_x.RS},ServerAlive message);
                AG_x.S sends(CS,State_AG_x message);
            else if (P_j==AG_x.RS) then
                AG_x.RS sends(P ∪ {AG_x.S},RepServerAlive message);
            else if (P not send(S, any message) in the last T ms ) then
                P_j send( S, PeerAlive message);
            end
        end
    end
end
```

Algorithm 3: Maintenance algorithm of each AG_x in the Distributed Area.

By means of applying the previous maintenance algorithm, the Server $AG_x.S$ and the Replicated Server $AG_x.RS$ can detect that any player P_{exit} has left the Auxiliary Game AG_x voluntarily or involuntarily. In such a case, the restructuring operation, described in Algorithm 4, is applied by the Server or the Replicated Server depending if the outgoing player was the Replicated Server or the Server, respectively. Therefore, the main aim of this algorithm is to replace the role of the outgoing player, Server or Replicated Server, by any other player of the Auxiliary Game. It means that if the $AG_x.S$ left the system, the $AG_x.RS$ would become the new $AG_x.S$ and a new $AG_x.RS$ will be searched among the rest of players of the AG_x. This is done by means of applying the same algorithm described in Algorithm 2. Thus, after applying the restructuring operation, independently of the player who has left the system, the AG_x is still alive for the rest of distributed players in a transparent way. Likewise, it is worth pointing out that the Auxiliary Game will be alive while any player $P_j \in P$ can assume the role of Server. Note that the cost of this algorithm is also $\theta(AG_x.size)$, where $AG_x.size$ is the number of peers of the AG_x.

```
Input: AG_x,P_exit;
Output: AG_x;
begin
    AG_x = AG_x − {P_exit};
    if (P_exit == AG_x.S) then
        AG_x.S=AG_x.RS;
    end
    if (AG_x.RS == ∅) then
        foreach (P_i ∈ AG_x) do
            AG_x.RS = AG_x.findServer(CRITERIA);
        end
        link(AG_x,AG_x.S,AG_x.RS);
    end
end
```

Algorithm 4: Player Exit Mechanism.

3.3 Analyzing the communication cost

Previous works described in Section 2 show that one of the main problems caused by the application of load balancing mechanisms in MMORPGs severs is the communication cost

provoked by the extra movement of players between servers. This section analyzes the communication overhead introduced by the application of the distribution mechanism of Algorithm 1. Due to the fact that the algorithm balances the load in a hybrid architecture, the communication overhead is calculated according on where it is applied, in the Central Area or in the Distributed Area.

3.3.1 Communication cost in the central area

In the case of distributing in the Central Area, we also calculate the overhead introduced by the classic method proposed by Bezerra et al. [6], described in Section 2, to be able to compare our proposal with a reference method of the literature based on distributing individual players among servers in a Client-Server system.

Table 1 shows the analytical evaluation of the communication overhead introduced by our architecture, named *Hybrid*, in relation to the method proposed by Bezerra et al., named *Classic*. This is calculated by the number of established communications between players and servers, excluding those caused by the game itself (for instance, the game state updates). For each method, we distinguish the communication overhead before applying the balancing method (*Before row*) and the overhead after balancing (*After row*). Finally, the last row in Table 1 indicates the total communication costs of both proposals.

Regarding to the Classic approach, before applying the balancing method, the number of players to be moved ($extra\_S\_players$) is multiplied by 2, as they have to communicate with both servers, the original and the destination one. Once players have been moved to the destination server, the Classic method introduces a communication overhead of $2 \cdot extra\_S\_players \cdot Interaction \cdot F$, where $Interaction$ is the percentage of players in the same AOI and managed by different servers, exchanging messages. The other parameter (F) indicates the frequency of this message exchange among players.

In the case of the Hybrid approach, before applying the balancing method, there will also be two communications for each player to be moved, which, in this case, is $extra\_S\_players$ plus 10% of max_S_load, according to our proposal in Algorithm 1. Thus, there is extra communication compared with the Classic method. However, after applying the balancing method, the additional communication cost is negligible, given that the balanced players belong to the same AG_x, which corresponds to their AOI and executed in the same server.

Therefore, Classic method introduces an overheard of $\theta(extra\_S\_players \cdot F \cdot Interaction)$ in front of the $\theta(extra\_S\_players)$ cost of the Hybrid method, which is much lower according to the typical values achieved by the *Interaction* and F parameters in a MMORPG game with low inter-player communication [33]. These results will be verified by simulation in the Section 5.1.

Table 1 Communication overhead of both load balancing proposals when distributing AG_x in the Central Area

	Classic (Players from MG)	Hybrid (Players from AG_x)
Before	$2 \cdot extra\_S\_players$	$2 \cdot (extra\_S\_players + max\_S\_load \cdot 10\ \%)$
After	$2 \cdot extra\_S\_players \cdot Interaction \cdot F$	0
Total	$2 \cdot extra\_S\_players \cdot (1 + F \cdot Interaction)$	$2 \cdot (extra\_S\_players + max\_S\_load \cdot 10\ \%)$

Table 2 Communication overhead of distributing AG_x to the P2P Area

	Hybrid (AG_x to P2P Area)
Before	$2 \cdot (extra\_S\_players + max\_S\_load \cdot 10\ \%)$
After	$(extra\_S\_players + max\_S\_load \cdot 10\ \%) \cdot Return_{RATE}$
Total	$(extra\_S\_players + max\_S\_load \cdot 10\ \%) \cdot (2 + Return_{RATE})$

3.3.2 Communication cost in the distributed area

In the case of balancing and managing the Auxiliary Games to the Distributed Area, the communication overhead is shown in Table 2. The cost related to the distribution of the Auxiliary Games to the P2P Area is the same as in the previous scenario ($2 \cdot (extra\_S\_players + max\_S\_load \cdot 10\ \%)$), given that the number of players to be moved is the same. When players are already distributed, the communication between CS and players during the Auxiliary Game is only due to the application of the Maintenance Algorithm, which is negligible given that the communication is only produced by the Server of each AG_x. When the Auxiliary Game is over, some of these Auxiliary Game players will return to the CS and, as a consequence, they will again communicate with the CS, obtaining $(extra\_S\_players + max\_S\_load \cdot 10\ \%) \cdot Return_{RATE}$, where $Return_{RATE}$ is the percentage of players returning to the CS. Section 5.2 evaluates this communication cost in relation to its main parameters.

4 Performance of the auxiliary game server

In this Section, given that our approach is based on using players' personal computers as the Server and Replicated Server for the Auxiliary Games in the Distributed Area, we analyze the viability in terms of CPU, memory and network consumption of current commercial desktops to fulfill server functionalities.

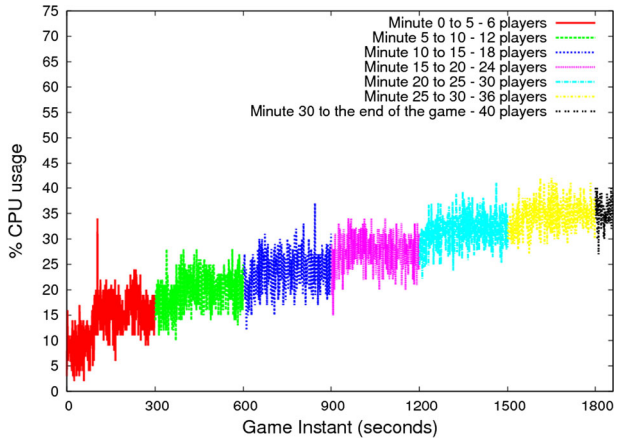

Fig. 3 CPU consumption in relation to the number of players

As our aim is to demonstrate that any current commercial desktop can be used for this purpose, this section is focused on analyzing the performance as a Server of a low cost desktop with the following features: Intel Core 2 Duo Processor at 2.2GHz with 2GB DDR2 SDRAM, 250GB SATA and an ADSL connection at 512Kbps.

In order to parameterize the game and to monitorize the consumption of computational resources by the Server, we played the *Urban Terror* game [18] increasing the number of players from 5 to 40 during the game. Although this game belongs to the MMOFPS category, its computational requirements and behavior are very similar to an MMORPG and the parameterization of the maximum number of concurrent players is easier. The open statistics [2] of PlaneShift [1] corroborate this assumption.

Figure 3 shows the percentage of use of CPU of the desktop computer while it is serving a game. The game was started with 5 players and every 300 seconds, 5 new players were connected until the game reached 40 concurrent players. It can be seen that there is a linear relationship between the CPU consumption and the number of concurrent players. This corroborates the results in Ye and Cheng in [37]. It is worth pointing out that taking the worst case of 40 concurrent players into account, the CPU consumption is always maintained below 45 %.

Figure 4 shows the memory consumption under the same conditions described above. Amplified in the center of the Figure, we can see how the server reserves a significant amount of memory at the beginning to initialize the game, but once it has begun, the memory utilization remains constant. Thus, our results reveal that the memory consumption is independent of the number of players. Likewise, we can see that the memory consumption is always below 100MB. Thus, we can conclude that memory is not a critical parameter for choosing a peer from the Distributed Area to act as a server.

The network requirements were also analyzed. Figures 5 and 6 depict the number of input and output packets to be transferred by the server when the number of concurrent players is increased from 5 up to 40 players over time. We can see the same trend as with the CPU case, where the number of input/output packets is roughly proportional to the number of players.

Finally, we analyzed the performance of a conventional ADSL connection of 512 kbps to serve a game party of 40 players. According to our empirical results, we can assume that:

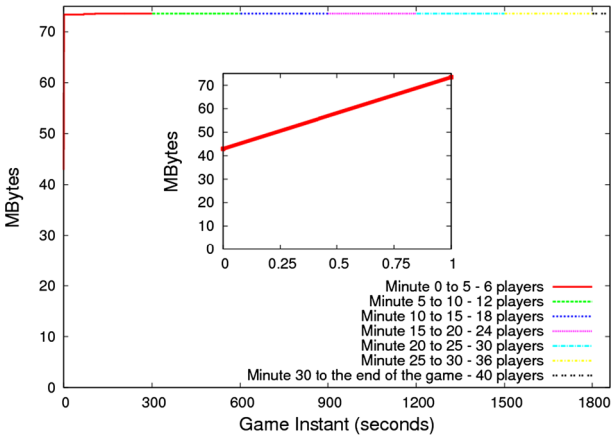

Fig. 4 Servers memory consumption in relation to the number of players

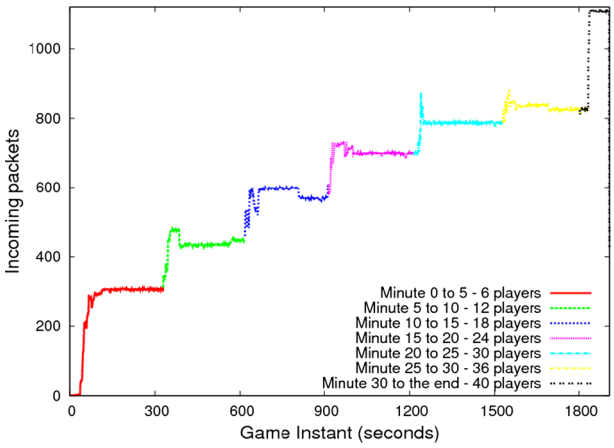

Fig. 5 Input packets in the server during the game

– Each output packet has an average size of 141 Bytes. Note that this value also fits with the empirical results shown in [16].
– The server sends two packets per second to each player in the game; this means 2,256 bps.
– Taking the size of the Auxiliary Game of 40 players into account, the average output bandwidth would be: $40\ players \cdot 2,256\ bps/player = 90,240$ bps.

Therefore, the output rate represents 17.2 % of the total capacity of the ADSL connection. This way, we can conclude that a current ADSL connection is enough to serve the connection of an Auxiliary Game.

The results shown throughout this section reveal that a commercial desktop can be used to serve a typical Auxiliary Game that usually has a size between 5 and 40 players, without any problem from the computational capacity point of view. Our experimentation has shown

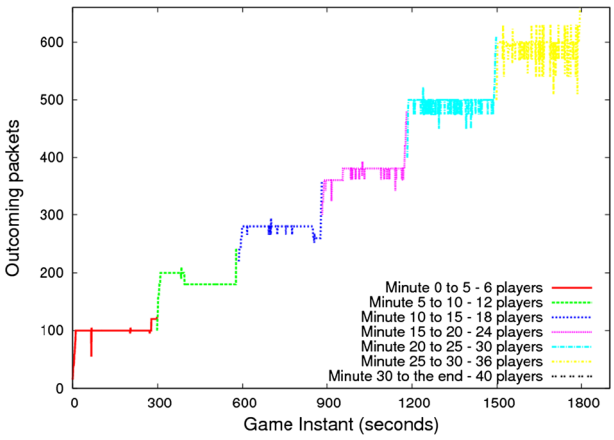

Fig. 6 Output packets in the server during the game

Table 3 Percentages of $AG_x.size$ modeled into the simulation

$AG_x.size$	5 players	10 players	20 players	25 players	40 players
Percentages (%)	66	14	2	3	15

that the most sensitive resource is the CPU, although it does not arrive in any case above 45 % of its performance.

5 Performance evaluation

In this Section, we evaluate the effectiveness of applying the load balancing mechanism to players in Auxiliary Games, considering them as an indivisible entity, and the scalability and viability of using the P2P area during the peak loads, that cannot be served by the cluster of servers. In addition, the QoS given to the Auxiliary Games players located in the P2P area is also analyzed in terms of latency and reliability.

The simulation was performed using SimPy [22]. SimPy is a discrete-event simulation language based on standard Python. SimPy tools were used to implement nodes of the platform, which can fulfill four distinct roles: player, $AG_x.S$, $AG_x.RS$ and CS. The SimPy procedures allow the random behavior of a player during the simulation to be represented. We carried out a set of 1,000 simulations, each of which consisted of a world game map whose dimensions were 1,000x1,000 (1 million player positions). This map was managed by a CS of 10 servers with a homogeneous maximum workload of 5,500 players per server. Taking into account the information about the existing game instances implemented in World of Warcraft [36], we modeled the Auxiliary Games with five different numbers of players: $AG_x.size = 5,\ 10,\ 20,\ 25$ and 40. Table 3 shows the percentage of each kind of AG_x in relation to the total number of AG_x. We considered that all the AG_x were scattered across the map using a random distribution that could eventually cause hot-spots. Figure 7 shows a snapshot of the game load map, where single players and different hot-spots caused

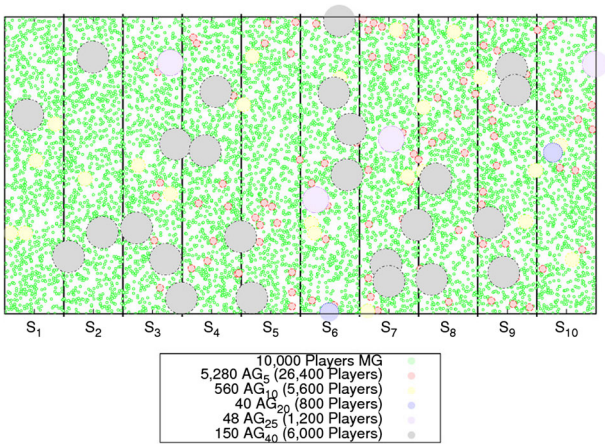

Fig. 7 Game load map snapshot

by the concentration of many AG_x of different sizes are plotted in different colors. Each rectangular region was managed by a single server in the central CS.

Another important issue is the calculation of the players' latency against the CS. This was determined by a triangulated heuristic, delimiting the 2-Dimensional Euclidean Space to ($x = [-1,000, +1,000]$, $y = [-1,000, +1,000]$). This methodology is based on the relative coordinates, explained in [14].

In order to find the best reliable player to act as a server, the understanding of the behavior of MMORPG players and their subsequent modeling is a key issue of our work. The modeling of players' uptime is based on their behavior history. Thus, we are able to predict how reliable a player will be in order to select the adequate ones to act as $AG_x.S$ and $AG_x.RS$ in terms of fault tolerance. From [35], we obtained mean and standard deviation values of players' uptime of 2.8 and 1.8 hours respectively. These values were taken from the World of Warcraft statistics as a representative player behavior for an MMORPG. These values can be modeled by a gamma distribution (see Fig. 8) with shape (K) equal to 1.157143 and a scale (Θ) equal to 2.419754, given that it has been successfully evaluated with the *Chi-square test*, indicating the goodness of our model in statistical terms. Then, with this model, we can assign a fault likelihood to any player taking as a reference the player' uptime that allows the server ($AG_x.S$) and its replicated ($AG_x.RS$) to be selected adequately for each Auxiliary Game (AG_x). It is a fault tolerance based model, which implies that despite the existence of extreme player behavior, in terms of sudden disconnections or large uptime, we prioritize the selection of those players with more stable connection settings. Thus, good reliability in the Distributed Area is ensured.

In order to test the proposed system and the load balancing mechanism, we have simulated the two following scenarios:

A In order to decrease the load average of servers progressively, the disconnection rate of players in the CS is 10 % higher than the connection rate. At the beginning of the simulation, the CS is loaded with its allowed maximum capacity of players: $max\_S\_load \cdot N$, where $N = 10$ is the number of servers of the CS. Moreover, the internal movements of players between servers are simulated; these movements are caused when players move across the game map. In this way, hot-spots arise dynamically in some servers while other servers are underutilized. Among all the players, 10 % are uniformly distributed across the Main Game map and the rest are located in the Auxiliary Games. This scenario is performed to show the differences between applying a load balancing mechanism to individual players, as has been usual in the

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

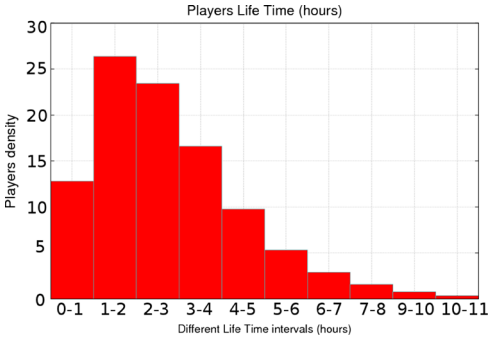

Fig. 8 Players uptime following a Gamma Distribution Histogram

literature, or taking the players' *AOI* into account that is to say, balancing players from the same Auxiliary Game as we propose in this work.

B In order to test the scalability performance of the system, the players' connection rate is 20 % higher than the disconnection rate. This means that the average load increases over the allowed maximum with time. In this scenario, the AG_xs not served by the CS due to overloading are sent to the P2P area. In addition, we simulate the number of players returning to the MG from the AG_x ($Return_{RATE}$), i.e., the percentage of players returning from the P2P to the Central Area, when the corresponding AG_xs are over. In this scenario, we want to prove that, thanks to the P2P area, it is possible to overcome the maximum load capacity of the Central Area, performed by a Client-Server architecture.

5.1 Testing load balancing methods

To analyze the effectiveness of the load balancing mechanism, explained in Section 3.2, we compared it with the load balancing approach proposed by Bezerra et al. in [6] and described in Section 2.

We applied both load balancing methods in scenario A, and the results are labeled *Classic* for the Bezerra method and *Hybrid* for our method. Figure 9 shows, for a CS with 10 servers (S_1, ..., S_{10}), the initial and final load of each server after applying both balancing methods. We can see that there were five overloaded servers at the beginning (S_4, S_7 , S_8 , S_9 , S_{10}); whereas the final distribution of players is balanced in both cases, thus avoiding overloaded servers. However, the Classic method lets more servers near the stress situation ($S_i.load \simeq 5,500$) due to the fact that movements are applied taking individual players as an entity, instead of taking Auxiliary Games as in our Hybrid method.

It is worth pointing out that the Hybrid method moves a significantly greater number of players, specifically 1,337 players in the Classic method versus the 4,100 players in the Hybrid method. This can be explained by two reasons. On one hand, movements affect all the players in each Auxiliary Game and, on the other hand, load balancing is applied to the extra-players ($extra\_S\_players$) plus 10 % of the maximum load of a server (max_S_load), with the aim of the load balancing mechanism being applied less frequently.

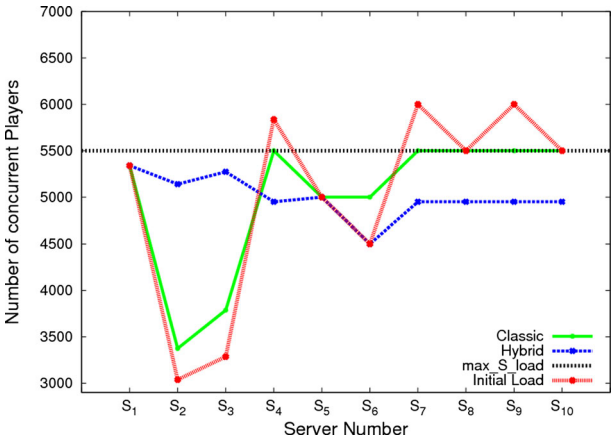

Fig. 9 Load balancing server comparison between Classic and Hybrid proposals for *Scenario A*

It is also worth remarking that this extra movement of players in the Hybrid method does not yield an increase in communication cost compared with the Classic method, because players in the same Auxiliary Game, who communicate intensively, are maintained in the same server after being balanced.

Figure 10 shows the communication cost described in Table 1 of Section 3.3, varying between 100 and 1,000 the value of $extra\_S\_players$. For the Classic proposal, the $Interaction$ parameter was set at 10 % and two different values of F, 20 and 25 messages per second, were evaluated. Note that according to the values shown in [33], the chosen $Interaction$ and F values correspond to an MMORPG game with low inter-player communication. Thus, the Classic method is favored over the Hybrid one. In spite of these unfavorable conditions, the Hybrid method exchanges many fewer messages than the Classic one, which corresponds with the analytical results obtained in the Section 3.3.

5.2 Scalability of the system

Scenario B was used to test the ability of the proposed Hybrid system together with the balancing method to scale on demand. Note that in this scenario, the load of each server of the CS exceeds the maximum allowed load; according to Algorithm 1, this is $max\_S\_load - 10\ \% \cdot max\_S\_load$, where $max\_S\_load = 5,500$ players. Under this scenario, for each server, Fig. 11 distinguishes the players located in such server and the players distributed from the server to the *P2P* area. From this Figure, we can see that the Hybrid proposal is able to exploit the P2P area distributing players and avoiding the overload situation. Thus, it is able to overcome the maximum number of players managed by a server, which is 4,950 players.

It is worth remarking that in our system the players' load of the Central Server would only increase in the hypothetical case that the players' connection rate was higher than the percentage of Auxiliary Games creation. According to the the nature and behavior of the players of the MMORPGs [33], more than 50 % of players of the game are playing in an Auxiliary Game and as a consequence, the system would achieve the saturation point in the hypothetical case that the players' connection rate was 50 % higher than the disconnection rate, which is a complete unusual situation.

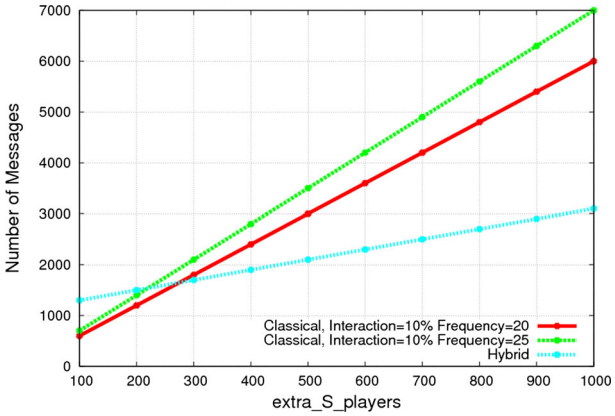

Fig. 10 Communication costs comparison for *Scenario A*

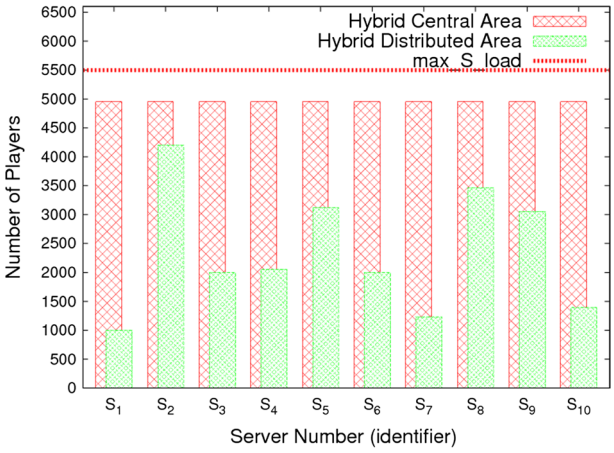

Fig. 11 Hybrid Distributed Area (*Scenario B*)

As we discussed in Section 3.3, a key aspect is the additional communication cost provoked by the balancing of players to the P2P area. Figure 12 depicts the evolution of the communication overhead showed in Table 2 for different values of the $Return_{RATE}$. In general, we can observe that whenever the $Return_{RATE}$ parameter increases, the volume of communications increases too, because more players have to communicate with the CS. However, it is remarkable that even in the worst case ($Return_{RATE} = 100$ % and $extra\_S\_players = 1,000$), the total amount of communications, assuming a message size equal to 141 Bytes [16], supposes an extra bandwidth of 719 KB for the CS, which is only slightly higher than a conventional ADSL connection.

Our evaluation demonstrates the viability of the Hybrid proposal for scaling the system on demand with a minimum additional communication cost for the cluster of servers. In order to guarantee the playability of the players in the Auxiliary Games distributed to the P2P Area, Quality of Service (QoS) mechanisms should be proposed. Accordingly, a QoS evaluation in the Distributed Area is analyzed in the next section.

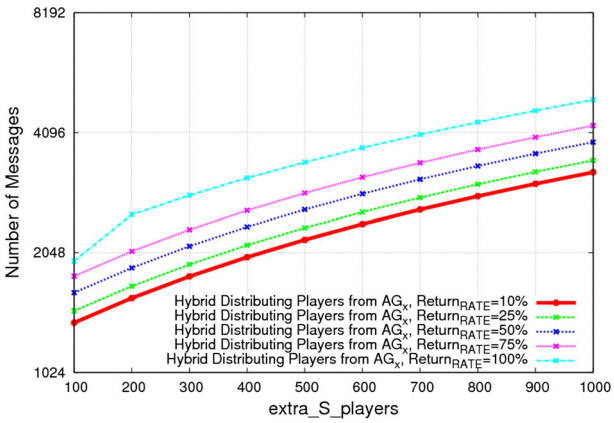

Fig. 12 Communication costs comparison for *Scenario B*

Table 4 *Distributes* function mechanisms performance under *LL* CRITERIA

CRITERIA	Latency Hybrid (ms)		Latency Client-Server (ms)	
LL	$AG_x.size = 5$	$AG_x.size = 10$	$AG_x.size = 5$	$AG_x.size = 10$
AVG	828.38	805.25	952.6	861.6
SD	3.132	5.620	3.5	6.1

5.2.1 QoS evaluation in the distributed area

The QoS of the proposed system indicates its ability to maintain latency values of the whole system under an acceptable threshold, while a huge number of players are managed on demand in the Distributed Area. The fault tolerance or reliability is also considered to be a parameter for measuring the QoS of the system. Thus, the system has to be able to minimize the sudden disconnections of players served in the Distributed Area.

It is worth pointing out that our system prioritizes latency or reliability depending on the chosen value as *CRITERIA* of the *distributes* function presented in Algorithm 2. Remember that the *distributes* function is used to find the optimal Server ($AG_x.S$) and Replicated Server ($AG_x.RS$) of each Auxiliary Game executed in the Distributed Area.

Table 4 shows the average (AVG) and standard deviation (SD) of the players' latency, when the latency (*LL*) CRITERIA is applied to distribute AG_xs over the Hybrid architecture in relation to the traditional Client-Server architecture, which means that players play directly against the CS instead of playing against the $AG_x.S$, as we propose. Note that AG_xs of size 5 and 10 players have been chosen as representative cases for Auxiliary Games.

Regarding the Hybrid architecture, the latency values are maintained for both AG_xs below 1 second, which is considered an acceptable threshold for *MMORPGs* [33]. In the case of the Client-Server architecture, we obtained an average latency time of 952.6 ms for $AG_x.size = 5$ and 861.6 ms for $AG_x.size = 10$, which represent increases in latency of 15 % and 7 % respectively compared with the P2P area. Note that the better latency time obtained in the P2P area is logical due that our method selects the players with the

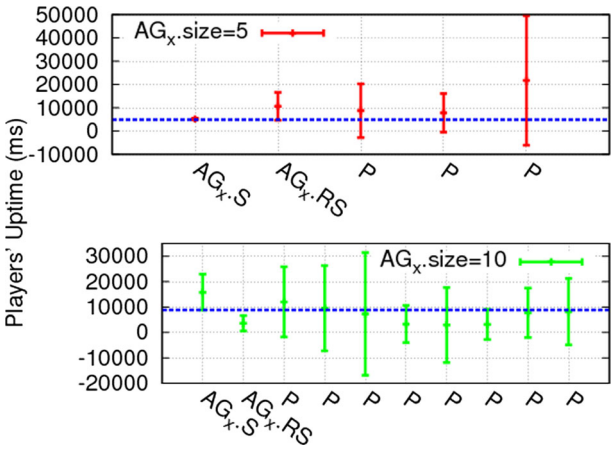

Fig. 13 $AG_x.S$ $AG_x.RS$ selection process for $AG_x.size = 5, 10$ players

Table 5 *PD* performance for $AG_x.size = 5, 10$

CRITERIA	$AG_x.size = 5$			$AG_x.size = 10$		
PD	$AG_x.S$	$AG_x.RS$	AG_x	$AG_x.S$	$AG_x.RS$	AG_x
Prob. of disconnection (%)	12	32	0.86	28	24	0.03

lowest latency in relation to the remaining players in the same AG_x, and this allows a local minimum to be obtained. Thus, the distribution based on AG_x in the P2P area is a key issue for providing scalability without latency penalization.

Regarding the appropriateness of the *PD* criteria for looking for $AG_x.S$ and $AG_x.RS$ servers of the AG_x, we studied its reliability. Figure 13 shows the predicted average uptime and the deviation of each player belonging to the $AG_x.size = 5$ and $AG_x.size = 10$, respectively. As a reference, the uptime of the chosen Server is depicted on the dotted line. From this reference line, we can see as always $AG_x.S.Uptime_{min} \geq P_j.Uptime_{min} | P_j \in AG_x - \{AG_x.S\}$.

In order to calculate the probability of disconnection (PD) of a player P_j, denoted as PD, we search the interval $[Uptime_x, Uptime_y]$ of the Gamma histogram, such as the given in Fig. 8, where $P_j.Uptime_{min}$ belongs to. Once the interval has been located, the PD value is calculated as the probability that $P_j.Uptime_{min} > Uptime_y$. For instance, given a $P_j.Uptime_{min} = 1.5\ h$, this will be located in the second interval $[1\ h, 2\ h]$ of the histogram of Fig. 8 and it will have a $PD(Uptime_{min} > 2h) = 60,9\ \%$. According to this, Table 5 shows the fault likelihood of the chosen $AG_x.S$ and $AG_x.RS$. In addition, it shows the fault likelihood of the whole AG_x by applying Algorithm 4 Given that the AG_x is alive while two players of the AG_x were playing, the failure probability is calculated as the product of the likelihood of $AG_x.S$, $AG_x.RS$ and the rest of players $P_j \in AG_x$, excluding the player with the worst reliability. As can be seen, in both cases, the fault likelihood of the AG_x is under 0.9 %. This enhances the fault tolerance of the mapping mechanisms. It is worth remarking that these percentages show the highest fault likelihood. Thus, it shows the worst performance case, which reveals the goodness of the *PD* mechanism combined with the role of the $AG_x.RS$.

6 Conclusion and future work

This paper confronts the problem of supplying computation service to the increasing demand by users to play in 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 on the players' wishes. In line with this, this paper proposes a hybrid architecture for playing MMORPGs. It is composed of a Central Area with a cluster of servers and a distributed P2P area that adds servers dynamically to the system according to the demand.

To distribute computation over this architecture, we defined a mapping mechanism that is based on moving Auxiliary Games, as an entity, among servers in the central cluster and the P2P area. This avoids a significant amount of communications in comparison of moving individual players, because players of the same Auxiliary Game have high interactivity during the game. By means of simulation, it has been demonstrated that the proposed mapping mechanism is able to provide well balanced loads in the cluster system while the distributed

platform scales on demand. Moreover, to maximize the performance in terms of latency and reliability, we proposed latency or reliability options to be used as *CRITERIA* for finding a new Server and Replicated Server for the Auxiliary Games in the Distributed Area. Likewise, we have shown that our proposal is able to achieve lower average latencies compared with the traditional Client-Server architecture. Concerning reliability, it has been demonstrated that the reliability mechanisms in our method is able to achieve a failure probability of less than 0.9 % in the worst cases.

Future work is oriented to face up the cheating problem related to MMORPG games in order to ensure a good player experience in the overall game. In order to exploit the peer's resources better, another route for improvement will be to manage the inherent heterogeneity of players. Finally, other interesting issue will be to merge our current mapping mechanisms with market criteria to reward the $AG_x.S$ and $AG_x.RS$, given that they are sharing their resources altruistically.

Acknowledgments This work was supported by the MEyC-Spain under contract TIN2011-28689-C02-02 and the CUR of DIUE of GENCAT and the European Social Fund.

References

1. Atomic Blue Corporation. Homepage of PlaneShift, the open source MMORPG. (http://www.planeshift.it/)
2. Atomic Blue Corporation. Link to PlaneShift server statistics, (http://v2.fragnetics.com/?page=gaming-serverlist)
3. Ahmed DT, Shirmohammadi S (2008) A Dynamic Area of Interest Management and Collaboration Model for P2P MMOGs. In: DS-RT '08: Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications. IEEE Computer Society
4. Assiotis M, Tzanov V (2006) A Distributed Architecture for MMORPG. In: Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames '06
5. Bauer D, Rooney S, Scotton P (2002) Network Infrastructure for Massively Distributed Games. In: NetGames
6. Bezerra C, Resin C (2009) A load balancing scheme for massively multiplayer online games. Multimed Tools Appl:45
7. Boulanger J-S, Kienzle J, Verbrugge C (2006) Comparing Interest Management Algorithms for Massively Multiplayer Games. In: Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames
8. Buyukkaya E, Abdallah Maha (2008) Data Management in Voronoi based P2P Gaming. In: Data Management in Voronoi based P2P Gaming, pp. 1050–1053
9. Castro M, Druschel P, Kermarrec A-M, Rowstron A (2002) SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE J Sel Areas Commun (JSAC) 20:1489–1499
10. chang Feng W, Chang F., chi Feng W, Walpole J (2002) Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server. In: Internet Measurement Workshop
11. Chen J, Baohua W, Delap M, Knutsson B, Honghui L, Amza C (2005) Locality Aware Dynamic Load Management for Massively Multiplayer Games. In: Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '05
12. De Vleeschauwer B, Van Den Bossche B, Verdickt T, De Turck F, Dhoedt B, Demeester P (2005) Dynamic Microcell Assignment for Massively Multiplayer Online Gaming. In: Proceedings of 4th ACM SIGCOMM workshop on Network and System Support for Games, NetGames
13. Douglas S, Tanin E, Harwood A, Karunasekera S (2005) Enabling Massively Multi-Player Online Gaming Applications on a P2P Architecture. In: Proceedings of the IEEE International Conference on Information and Automation

14. Eugene TS, Zhang H (2001) Predicting Internet Network Distance with Coordinates-Based Approaches. In: INFOCOM
15. Fan L, Trinder P, Taylor H Design Issues for Peer-to-Peer Massively Multiplayer Online Games. International Journal of Advanced Media and Communications, 4, 2010
16. Feng W-c, Chang F, Feng W-c, Walpole J (2002) Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment, IMW '02, pages 151–156
17. Fengyun L, Parkin S, Morgan G (2006) Load Balancing for Massively Multiplayer Online Games. In: Proceedings of 5th ACM SIGCOMM workshop on Network and System Support for Games, NetGames
18. Frozesand. Homepage of Urban Terror, the open source MMOFPS
19. Hampel T, Bopp T, Hinn R (2006) A Peer-to-Peer Architecture for Massive Multiplayer Online Games. In: Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames
20. Hanzich M, Hernandez P, Gine F, Solsona F, Lerida JL (2011) On/off-line prediction applied to job scheduling on non-dedicated NOWs. J Comp Sci Technol 26(1):99–116
21. Huang G-Y, Shun-Yun H, Jiang J-R (2008) Scalable reputation management with trustworthy user selection for P2P MMOGs. Int J Adv Media Commun 2:380–401
22. IBM Developers Works. Charming Python: SimPy Simplifies Complex Models (Simulate Discrete Simultaneous Events for Fun and Profit). 2002
23. Keller J, Solipsis GS (2003) A Massively Multi-Participant Virtual World. In: PDPTA
24. Knutsson B, Honghui L, Wei X, Hopkins B (2004) Peer-to-Peer Support for Massively Multiplayer Games. In: INFOCOM
25. Lee YT, Chen KT (2010) Is Server Consolidation Beneficial to MMORPG? A Case Study of World of Warcraft. In: Proceedings of IEEE International Conference on Cloud Computing
26. Liu H, Lo Y (2008) DaCAP - A Distributed Anti-Cheating P2P Architecture for Massive Multiplayer On-line Role Playing Game. In: CCGRID
27. Liu H-I, Lo Y-T (2008) Dacap - A Distributed Anti-Cheating Peer to Peer Architecture for Massive Multiplayer On-line Role Playing Game. In: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid
28. Matsumoto N, Kawahara Y, Morikawa H, Aoyama T (2004) A Scalable and Low Delay Communication Scheme for Networked Virtual Environments . In: Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE, pages 529–535
29. Research in China. China Online Games Market Report, 2008. http://www.researchinchina.com/Htmls/Report/2008/1944.html
30. Rhalibi AE, Merabti M (2006) Interest Management and Scalability Issues in P2P MMOG. In: Consumer Communications and Networking Conference, (CCNC) 3rd IEEE, vol 2, pp. 1188–1192
31. Siu Fung Y (2006) Hack-Proof Synchronization Protocol for Multi-Player Online Games. In: Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames
32. Storey K, Fengyun L, Morgan G (2004) Determining Collisions between Moving Spheres for Distributed Virtual Environments. In: IEEE Proceedings of the Computer Graphics International, pages 140–147
33. Suznjevic M, Dobrijevic O, Matijasevic M (2009) MMORPG player actions: Network performance, session patterns and latency requirements analysis. Multimed Tools Appl:45
34. Swamynathan G, Zhao BY, Almeroth KC (2008) Exploring the Feasibility of Proactive Reputations: Research Articles. Concurrency Computation: Practice Experience 20:155–166
35. Tarng P, Chen K, Huang P (2008) An Analysis of WoW Players Game Hours. In: NetGames
36. WoWWiki. Your Guide to the World of Warcraft, 2011. http://www.wowwiki.com/Instances_by_level
37. Ye M, Cheng L (2006) System-Performance Modeling for Massively Multiplayer Online Role-Playing Games. IBM Syst J 45:45–58
38. Yu AP, Son TV (2005) MOPAR: A Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games Online Games, NOSSDAV?05, ACM. ACM Press

Ignasi Barri received the B.S. in computer science engineering from the Universitat of Lleida (UdL), Spain, in 2009 and Ph.D degree in computer science from the UdL, Spain, in 2012. Actually, He is combining his current job position as an International Research Project Manager in the multinational Indra with teaching engineering to the University of Lleida. His research interest includes peer-to-peer computing and MMOG games.

Concepció Roig received the MS and PhD degrees in Computer Engineering from the Universitat Autnoma de Barcelona, Spain, in 1996 and 2002 respectively. She has been an associate professor in the Department of Computer Science at the University of Lleida, Spain, since 1992. Her current research interests include parallel and distributed systems, modelling of parallel applications, task graph models, static mapping and optimization of parallel applications.

Francesc Giné received the B.S.in telecommunication engineering from the Universitat Politecnica de Catalunya (UPC), Spain, in 1993 and the M.S. and Ph.D degrees in computer science from the Universitat Autnoma de Barcelona (UAB), Spain, in 1999 and 2004, respectively. He is currently an Associate Professor of Computer Arquitecture at University of Lleida (UdL), Spain. His research interest includes multicluster and peer-to-peer computing and scheduling-mapping for parallel processing.