Peer-to-Peer Networking and Applications manuscript No.
(will be inserted by the editor)

Integrating Peer-to-Peer and Cloud Computing for Massively Multiuser Online Games

Hanna Kavalionak · Emanuele Carlini · Laura Ricci · Alberto Montresor · Massimo Coppola

Received: date / Accepted: date

Abstract Cloud computing has recently become an attractive solution for massively multiplayer online games, also known as MMOGs, as it lifts operators from the burden of buying and maintaining large amount of computational, storage and communication resources, while offering the illusion of infinite scalability. Yet, cloud resources do not come for free: a careful orchestration is needed to minimize the economical cost. This paper proposes a novel architecture for MMOGs that combines an elastic cloud infrastructure with user-provided resources, to boost both the scalability and the economical sustainability provided by cloud computing. Our system dynamically reconfigures the platform while managing the trade-off between economical cost and quality of service, exploiting user-provided resources whenever possible. Simulation results show that a negligible reduction in the quality of service can reduce the cost of the platform up to 60% percent.

Keywords Massively Multi-player On-line Games · Cloud Computing · Peer-to-Peer · Distributed Systems

1 Introduction

In the last years, on-line gaming entertainment has acquired an enormous popularity among both industrial and academic researchers. This attention is justified by the economic growth of the field, in particular regarding *massively multiplayer online games* (MMOG). According to [51], in 2010 the MMOG market was worth 6 billion dollars worldwide with a predicted value of 8 billion dollars by 2014, whilst the number or users reached 20 million users worldwide in 2010[1].

MMOGs are large-scale applications providing a real-time, shared, persistent and seamless *virtual environment* (VE) to huge communities of users. MMOGs operators provide the necessary hardware infrastructure to support such communities, obtaining their profit from the fees users pay periodically to access the game. Therefore, operators' profit is strictly linked to the number of users who participate in the MMOG; moreover, the more popular is a game, the more attractive it becomes for new users. For this reason, offering an acceptable level of service while assuring the infrastructure to sustain a large number of users is a core goal for MMOG operators.

Today's architectures for MMOGs rely on a client/server model. This centralized approach enables a straightforward management of the main functionalities of the virtual environment, such as user identification, state management, synchronization among players and billing. However, with larger and larger numbers of concurrent users, centralized architectures are hitting their scalability limits, especially in terms of economical return for the operators.

Virtual data locality has been used to increase scalability of MMOG. The most common approach, called *zoning* [31], divides the VE into regions, each region independently managed by a server. As in zoning, *mirroring* [16] divides the VE into regions, but a region can be replicated in multiple servers at the same time. A infrastructure is then dedicated for the consistency of the mirrored regions. *Instancing* [4] is similar to mirroring, but each replica of a region is independent, such that players cannot communicate among different replicas.

Kavalionak H., Montresor A.
University of Trento
E-mail: {alberto.montresor,hanna.kavalionak}@unitn.it

Carlini E., Coppola M.
Institute of Information Science and Technologies CNR-ISTI, Pisa, Italy E-mail: {emanuele.carlini,massimo.coppola}@isti.cnr.it

Ricci L.
University of Pisa E-mail: ricci@di.unipi.it

[1] `http://www.mmodata.net`, August 2012

https://doi.org/10.1002/cpe.3289

These three mechanisms help distribute the load of the MMOG on multiple servers. However, they do not address the problem of resource provisioning. Indeed, server clusters have to be bought and operated to withstand service peaks, also balancing computational and electrical power constraints. A cluster-based centralized architecture concentrates all communication bandwidth at one data center, requiring the static provisioning of a large bandwidth capability. This may lead to *over-provisioning*, which leaves the MMOG operators with unused resources when the load on the platform is not at its peak.

On-demand resources provisioning (also known as cloud computing [6]) may alleviate the aforementioned scalability and hardware ownership problems [39,41]. The possibility of renting machines lifts the MMOG operators from the burden of buying and maintaining hardware, and offers the illusion of infinity resource availability, allowing (potentially) unlimited scalability. Also, the pay-per-use model enables to follow the daily/weekly/seasonal access patterns of MMOGs.

However, the exploitation of cloud computing presents several issues. The recruiting and releasing of machines must be carefully orchestrated in order to cope with start-up times of on-demand resources and to avoid incurring on unnecessary expenses due to unused servers. Further, besides server time, bandwidth consumption may represents a major expense when operating a MMOG. Thus, even if an infrastructure based entirely on on-demand resources is feasible, the exploitation of user-provided resources may further reduce the server load and increase the profit margin for the MMOG operator.

These aspects have been investigated extensively in the research community in the last decade [5,26,20]. Mechanisms to integrate user-provided resources in a MMOG infrastructure naturally evolved from the peer-to-peer (P2P) paradigm. By reducing the load on centralized servers, P2P-based solutions present several attractive advantages. First, P2P techniques are inherently scalable – the available resources grow with the number of users. Second, if a peer fails, P2P networks are able to self-repair and reorganize, hence providing robustness to the infrastructure. Third, network traffic is distributed among the users involved, making difficult the creation of bottlenecks. Furthermore, all these properties pair with little or no costs for the VE operators.

However, P2P-based infrastructures require additional mechanisms to suit the requirements of a MMOG. When a peer leaves the system, its data must be transferred somewhere else; given that the disconnection may be abrupt, replication mechanism must guarantee that data will not be lost. The lack of a central authority hinders security and anti-cheating enforcement. Moreover, user machines typically have heterogeneous constraints on computational, storage and communication capabilities, making them complex to be exploited.

The high degree of complementarity between on-demand and user-provided resources suggests to strictly integrate the two approaches. The core idea is to allow operators to choose the balance between on-demand and user-provided resources. In our design, an operator can decide to have an infrastructure more reliable and responsive (for example for particularly interactive MMOGs) or to reduce the economical effort and provide a less powerful infrastructure, perhaps suitable for less interactive MMOGs. In other words, the idea is to let the operator to choose on how to make profit, either by offering a more controllable service, or by saving on the cost of infrastructure or in a point in the middle between the two.

These considerations drove the design of the MMOG architecture presented in this paper. Our contribution can be outlined as the following. First, our infrastructure for the management of MMOG objects is based on a Distributed Hash Table (DHT). Our DHT exploits *virtual nodes* [23]. Each virtual node corresponds to a contiguous and non overlapping portion of address space of the DHT. Each virtual node contains a set of MMOG objects and it can be dynamically associated to physical machines with a little disruption in the underlying DHT structure. This allows us to migrate objects among cloud- or user-provided resources with few impact on the QoS. Second, we developed a dynamic resource provisioning model, that manages the migration of the objects between physical machines according to the preferences defined by the operator. The provisioning approach allows to cope with the impact caused by the start-up times and over-renting of cloud on-demand resources. Third, we provided a model for the QoS and the economical cost of the MMOG on top of the integrated architecture. Following our model, a MMOG operator can tune the system according to the application requirements. Finally, we tuned and evaluated the architecture through extensive simulations. The results showed that slightly decreasing the quality of service may yield a cost reductions up to 60%.

The paper is structured as follows. Section 2 discusses the architectural background that puts our contribution into context. Section 3 provides the definition of the system model, whereas Section 4 presents the algorithm for resources orchestration in details. Section 5 evaluates the platform through simulations. Section 6 offers an overview of the related work, with particular focus on hybrid MMOG architectures and P2P/cloud computing integration. Finally, Section 7 concludes the paper.

2 Architecture

This section introduces the overall structure of the proposed MMOG hybrid architecture. In order to motivate our design

choices, we first briefly review the main characteristics of a centralized single server architecture for MMOGs.

Players connect to a centralized server by means of a *game client*. The game client shows on the player's screen the visual representation of the virtual environment and maps the actions of the player (i.e. movements and/or interactions with objects) into communications with the server. The virtual environment is populated with *entities*, which can be *avatars* representing players or *objects* that can be manipulated. The player actions can be classified as *positional actions* and *state actions* [28]. State actions correspond to changes of the entity's state, e.g. the act of closing a door or collecting an object. Positional actions correspond to the movement of entities across the virtual environment and are the ones that change the positions of the avatars. The positional actions may trigger a *migration* if the DVE is geographically divided into zones and those zones are assigned to the nodes. Since a node manages the zone and all its content (including avatars), when an avatar changes zone, it should also be transferred to another node.

When an action occurs, the server must spread the information to the other players. However, not all players are interested in all actions. In fact, players only receive actions executed inside an area centered at the their virtual position, called *area of interest* (AOI). It is a task of the server to dynamically update and maintain the AOI of the players.

2.1 Distributed MMOG

Unlike single server architectures described above, distributed MMOG architectures employ strategies to divide the virtual world into *regions*, and to find a proper assignment of these regions to multiple *nodes*. We use the generic term "node" to indicate either peers running by the user, or virtual machines running inside a cloud. Some architectures adopt a spatial division of the virtual environment into regions, and assign all the entities in a region to a node. Region-based partitioning is efficient for AOI resolution; since entities are clustered according to their spatial position, identifying entities included in an AOI is a relatively easy task. However, entities distribution is usually not spatially uniform, due to the presence of *hotspots*, i.e. regions with high concentration of entities. One of the drawback of hotspots is that they generate a large amount of load on the nodes managing them. Furthermore, due to the spatial division, positional actions may trigger a change into entity-region assignment, implying the transfer of entities among nodes. This may reduce the interactiveness of the game, given that entities are not accessible during transfers, in fact denying any possible state action on it. This is even more critical when considering the rate of transfer that in turn depends on the rate of positional actions (usually high) and the dimension of the regions.

By comparison, an hash-based entity assignment presents complementary characteristics. Since the association of an entity to a node does not depend on the position of the entity, positional actions do not trigger any migration of objects among nodes. Also, due to the random assignment, entities in a hotspot are managed by several nodes, whose load is uniformly distributed. However, this solution makes AOI resolution impractical (the objects of an AOI may be spread among different nodes) and therefore it is rarely used in practice.

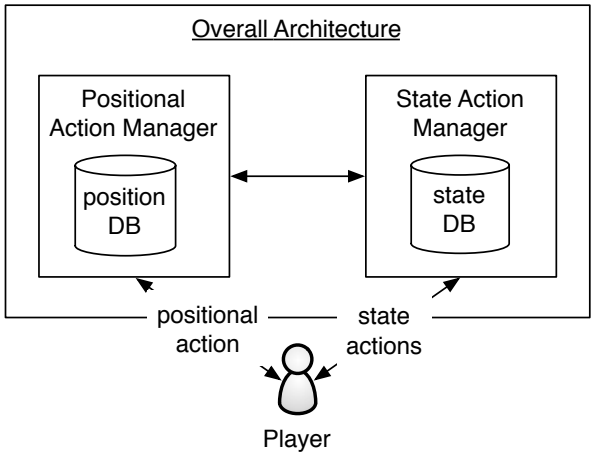

Fig. 1 Overall architecture

2.2 The proposed architecture

In order to retain the advantages of both the entity assignment strategies discussed above, we propose a distributed MMOG architecture (shown in Figure 1) that exploits two components, each one managing a different kind of actions. The *positional action manager* (PAM), which we previously presented in [10]), manages the positions of the entities by organizing a epidemic-based distributed overlay among players. The *state action manager* (SAM), which is the focus of this paper, stores the entities state and is organized according to an entity-to-node assignment based on hashing. This assignment strategy enables to handle the state of the entities without any transfer of them across nodes due to positional actions. Such transfers may anyway occur, but instead of being triggered by positional actions, they are usually performed to optimize the distribution of the entities (and as a consequence, of the load) among the nodes.

2.3 State action manager

In order to build and maintain an overlay for the management of the entity state in the MMOG, the state action manager (see Figure 2) is based on a distributed hash table (DHT)

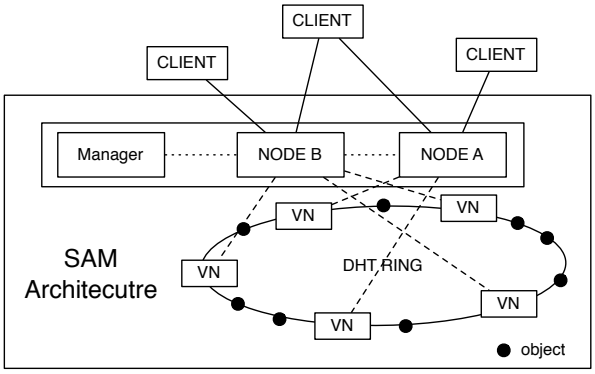

Fig. 2 Black dots are the objects inside the virtual environment. VN boxes correspond to virtual nodes. Node *A* manages 2 VNs, whereas node *B* manages 3 VNs. Client connects to the nodes to modify and read the objects. The manager has a global knowledge of the state of the node and the VNs.

[47,46]. A typical DHT manages a logical address space, whose size is large enough to avoid clashes among items (i.e. a common size is 2^{160}). Each entity of the MMOG (avatar, objects) is assigned with an address in such space, which we refer to as its ID. The IDs are uniformly assigned to balance the distribution of the entities in the address space. The address space is partitioned among the nodes, together with the associated entities. Nodes are then connected to each other by an overlay, for routing and synchronization purposes. The overlay is built to guarantee $O(\log N)$ bounds, where N is the number of nodes in the DHT, both for the routing hops and for the size of the routing tables.

In addition to the typical DHT mechanisms, we adopt the *virtual node* (VN) paradigm over DHTs proposed by Godfrey et al. [23], to introduce a clear separation between the logical and the physical nodes. Each virtual node is in charge of an address range of the DHT. Several virtual nodes may be allocated on the same physical node. From a client perspective, a virtual node acts as a state server for a set of entities. Since the entities that a client is interested in may be managed in principle by different virtual nodes, each client may have multiple simultaneous connections to them. For instance, in Figure 2, a client is connected with node *A* and *B* at the same time. Therefore, the number of concurrent connections for a player is then bounded by the amount of entities in its AOI. Also, entities are normally not updated at the same time altogether. Hence, even if a connection exists between a player and a VN, it is used only when entities are actually modified and VN pushes the updates.

In our architecture, we define the *load* of a VN as the upload bandwidth consumed to broadcast entities state to its associated clients. The load depends on the amount of entities that correspond to the VN and the amount of clients accessing them. The load changes over time, according to the interaction pattern of the avatars. Moreover, load may be unbalanced due to the presence of more popular entities. For instance, objects belonging to an hotspot receive an higher amount of updates.

The proposed architecture also includes an additional module, called *manager*, whose goal is to distribute the load among the nodes, so to exploit their heterogeneity. DHT nodes periodically notify the manager with their own load information. The manager periodically computes new assignments node-VNs based on the received information and, if necessary, the enrollment or the disposal of nodes from the DHT. The issue of the manager placement is out of the scope of this work. We consider the manager to be centralized and statically placed on a proprietary server. In this scenario, the adoption of the VN paradigm yields concrete advantages: (i) more powerful nodes may receive a higher number of VNs than less powerful ones, (ii) heavy loaded nodes may trade VNs with unloaded ones, (iii) in case of a physical node failure, its VNs are possibly transferred/reassigned to different, unloaded, physical nodes, so reducing the risk of overloaded nodes. Moreover, migrating VN is easy and light. Their migration does not affect the organization of the address space at the DHT level. It only requires the exchange of data managed by the VN as well as the update of the mapping between the logical identifier of the VN and the physical address of the node hosting it.

2.4 NAT-traversal

A relevant limitation of deploying real P2P systems on the Internet is the fact that a large part of all the nodes are behind Network Address Translation (NAT) gateways and firewall systems. This kind of architecture makes it difficult for two nodes belonging to different private sub-networks to contact each other directly. At the same time, on-line gaming type applications require nodes (i.e. players) to be able to communicate directly. NAT traversal issues are well known in the field of P2P communication and widely addressed in the literature [19,44]. We leave the development of the NAT traversal approach for our architecture as future work and in the following we assume that nodes are able to communicate directly.

2.5 Virtual nodes

One of the main advantages of the virtual node approach is the possibility to easily move entities across the nodes of the DHTs. This ability is a fundamental requisite for enabling proactive load distribution mechanisms. To better understand the advantages of exploiting virtual nodes, let us spend a few words on the load distribution in classical DHTs (i.e. that does not employ virtual nodes). There are essentially two ways to dynamically distribute the load in classical DHTs:

1. *Move nodes*. An unloaded node (i.e. A) moves to a precise address of the DHT, so to unload a heavy loaded node (i.e. B). This operation requires A to leave the DHT and rejoin in a position so that part of the load form B is transferred to A. Even if this approach may work in general, it is too time consuming and creates too overhead for a real-time application as a virtual environment. To fully understand the process, let us consider C as the successor of A (i.e. the node that is after A in the ring-shaped space of the DHT)[2]. Let us suppose that the DHT includes a node B which is overloaded, while C is underloaded. A can leave and then re-join the DHT in a position preceding B in order to unload B. When A leaves, C becomes responsible of the address space left free by A. This information must be spread in the DHT, so that the routing for the former A address space points correctly to C. In addition, before leaving, A must transmit all the data on its entity descriptors to C. When A joins the DHT and becomes the predecessor of B, this information must be spread to the DHT to adjust routing path. B also must send to A the entity descriptors that are in the new address space of A. In addition, A must build its routing table, in order to be part of the overlay. In summary, this process requires the transfer of two set of entities (from A to C and from B to A), the spreading of new information about 3 nodes and the building of a new routing table. All these operations take time, and, most important, imply a large number of transferred data during which the entities are not reachable from clients.
2. *Move descriptors*. This technique requires moving the entity descriptors among nodes to distribute the load. Practically, an entity descriptor changes its ID in the ring-shaped address of the DHT. During the transfer of the descriptor, the entity is not accessible by clients. However, moving descriptors prevents clients the ability of caching the address of the descriptors, since any time a client accesses to an entity, it must query the DHT for its position. This requires to wait $O(\log N)$ steps, which may be too long with an high number of nodes.

With virtual nodes, load distribution is lighter and more flexible with respect to a classical DHT. Node directly exchange virtual nodes (a process that we call *virtual node migration*), which offers the following advantages:

- the ID of the entity does not change over time;
- it is possible to transfer load without nodes to leave the DHT;
- a virtual node that has moved does not have to rebuild its entire routing table. In fact, moving a virtual node requires only to stabilize a few routing paths, much less than in a classical DHT system;
- it is possible to partially increase or decrease the load of a node.

During a VN migration, entities of the VN cannot be accessed. In other words, players cannot interact with the objects inside the VN that is migrating. Also, it can be the case of a player modifying the state of the object locally, just to see it reverted back when the migration of the corollary is completed. To this end, it is important to keep the transition time as short as possible, in order to provide an acceptable level of interactivity for the VE clients.

2.6 Replication and fault tolerance

In a distributed system, the need of replication comes from the intrinsic unreliability of nodes. Since we target an heterogeneous system including both peer and cloud nodes, a fair orchestration of replication is a relevant issue. Our approach is based on the reasonable assumption that, in general, cloud nodes can be considered *reliable* whereas peer nodes are *unreliable*, due to the high degree of churn which characterizes P2P systems. This difference is mainly due to the lack of control over peers, which are prone to unexpected failures, and may leave the system abruptly. On the other hand, cloud nodes generally belong to a stable infrastructure based on virtualization, and this greatly increases their robustness and flexibility.

In order to cope with the unreliability of peers, we propose that every VN assigned to a peer is always replicated. The replica, called *backup virtual node* (bVN), is then assigned to a trusted resource, i.e. a cloud node. To keep the state of the bVN up-to-date with the original, peers send periodic updates to the cloud nodes. The replica schema adopted is *optimistic* [45], i.e. players can access to entities without previous synchronization between the regular VN and the relative bVN. This schema leads to *eventual consistency*, favoring availability over consistency of the entities. The periodic updates from the peer to cloud for synchronizing bVN add further bandwidth requirements. However, the synchronization is performed at relatively large intervals (e.g. 30 seconds) with respect to the player updates, to reduce the required bandwidth.

The presence of bVNs guarantees a certain degree of availability in case of peer failures. Consider a peer P that manages a single VN and cloud node C that manages the respective bVN. When P departs from the system, either abruptly or gracefully, C becomes the new manager of the primary replica. As a consequence, clients connected to P must now connect to C. In the case of a gracefully departure of P, P itself may inform them about the new role of C;

[2] We consider Chord in this example, but with small differences the following considerations are valid for other DHT implementations as well

Δt	length of a time step
VN	set of virtual nodes
$v_{load}(t)$	upload bandwidth load in byte of v at time t
$v_{ent}(t)$	entities managed by v at time t
$N(t)$	set of nodes at time t
n_{cap}	capacity of n
n_{bcost}	upload bandwidth cost per byte of n
n_{rcost}	renting cost of N at t
$n_{load}(t)$	upload bandwidth load in byte of n at time t
$n_{lf}(t)$	load factor of n at time t
$n_{vs}(t)$	the set of virtual nodes managed by n
$n_{pl}(t)$	player being served by n at time t
$\alpha(t)$	$[0,1]$ QoS of the platform at time t
α_{thres}	QoS threshold
$\beta(t)$	cost in USD of the platform at time t
$DU(\Delta t)$	delayed updates during Δt
$U(\Delta t)$	total updates during Δt
z	number of updates per second
$E[latency_{i,j}]$	expected latency between nodes i and j
$E[fail_n]$	expected failure probability for node n

Table 1 Table of symbols

otherwise, the involuntary departure of P can be detected either by C, since it receives no more updates from P, or from the DHT neighbors of P, due to the repairing mechanism of DHTs. These nodes are able to notify the clients to send their notification to C.

3 System model and problem statement

In this section, we provide a model of our distributed system and introduce the problem statement; in order to improve the readability, in Table 3 lists the parameters used here. Let time be subdivided in discrete time steps of length Δt and denoted by $t \in \mathbb{N}$. Let VN be the set of virtual nodes in the system. Then, $\forall v \in VN$ we define $v_{load}(t)$ as the bandwidth consumed by the virtual node v in the time step t, with $v_{load}(0) = 0$. Similarly, $\forall v \in VN$ we define $v_{ent}(t)$ as the number of entities managed by v at time t, with $v_{ent}(0) = 0$. Each virtual node in VN is assigned to a node.

The set $N(t)$ contains the nodes that are in the system at the time t. An arbitrary node $n \in N(t)$ is characterized by the following invariant properties: (i) bandwidth capacity n_{cap}, (ii) bandwidth cost n_{bcost}, (iii) renting cost n_{rcost}. Over time, nodes are assigned with virtual nodes. We indicate with $n_{VN}(t)$ the set of virtual nodes assigned to a node n at time t. The outgoing bandwidth load imposed on a node at time t is indicated as $n_{load}(t) = \sum_{v \in n_{VN}(t)} v_{load}(t)$. From this, we define $n_{LF}(t) = n_{load}(t)/n_{cap}$ as the load factor of n at t. Finally, we indicate with $n_{pl}(t)$ the number of players served by n at time t.

We consider two different kinds of nodes, virtual machines rented from a cloud and peers provided by users. User-provided resources have no associated cost for bandwidth and renting, while cloud nodes are assigned with a pricing model taken from Amazon EC2 [1], with the assumption that we can charge cloud nodes per unit of time Δt. Hence, it is possible to compute the cost per time unit as the sum of the bandwidth cost and the renting cost of the nodes, according on the bandwidth consumed at time t. The total system cost $\beta(t)$ is computed as follows:

$$\beta(t) = \sum_{n \in N(t)} ((n_{load}(t) * n_{bcost}) + n_{rcost}) \quad (1)$$

In this formulation the cost due the upload bandwidth changes over time. Rather, the renting cost depends on the number of cloud nodes exploited.

3.1 Quality of service

Each virtual node offers a service that is comparable to a publish/subscribe system [25]. Players subscribe to nodes, send inputs and receive back updates at a fixed rate. The specific rate depends on the particular MMOG genre, typically in the order of few updates per second. Here we define this frequency as $z = 4$, corresponding an update every 250ms which fits medium-paced MMOGs [15]. When updates are delayed, players may perceive a clumsy interaction with the virtual environment. If the number of consequent delayed updates is large, players might not be able to interact with the environment at all. To favor a fully interactive virtual environment, the rate of updates should be as stable as possible. We consider the capacity of the nodes to provide a constant rate of updates as the metric for the *Quality of Service* (QoS).

In general, there are two main causes for delayed updates: (i) the network infrastructure between the node and the client, and (ii) the ability of the node to send the updates in time. In the first case, since we assume the Internet as the communication media, latency spikes and jitter are responsible of delayed updates. In this paper we do not consider this issue, since it is general for any architecture. Moreover, several solutions (such as LocalLag [35]) have been proposed to mitigate the effects of network delays in MMOGs.

In this paper we consider the second case (i.e. the delays generated by servers), as it is greatly affected by the exploitation of user-provided resources. In fact, peers are more prone to delay updates rather than a datacenter server, given their smaller reliability and limited bandwidth capability.

As QoS measure, let $\alpha(t) \in [0,1]$ be the fraction of updates the nodes send within time t for all the entities whose state changed at time $t-1$. For example, if at a given time t the nodes successfully sent only half of the updates for all the entities whose state is changed at time $t-1$, then $\alpha(t) = 0.5$. Let us define $U(t)$ as the total number of updates from $t-1$ to t and $DU(t)$ as the number of delayed updates from $t-1$ to t. Then:

$$\alpha(t) = 1 - \frac{DU(t)}{U(t)} \quad (2)$$

There are three cases in which the nodes might incur in delayed updates:

- *Virtual node migration*. During a migration, the service is unavailable for the time the entities are transferring between nodes. In this case, the number of delayed updates depends on the *migration time* (MT, the times for a VN to migrate between nodes), which is discussed in Section 5.2.
- *Overloading*. When a node is overloaded, it simply does not have enough bandwidth to send updates. As a consequence, it either drops or delays some updates.
- *Failures*. When a node crashes and a back up node takes its place, the players need time to "know" the new updates provider. During this time the players are not receiving new system updates.

In a sense, migrations and overloading can be seen as a "necessary evil", as they trade some QoS in exchange of more flexibility. Hence we do not take them into account when we compute the target QoS for virtual node assignment. The migration is a graceful process that we can tune to minimize the affection on the QoS. In particular, the size of the virtual nodes can be chosen so that their MT remains under a definite time threshold. The details of this aspect and the tuning are discussed in Section 4.3.

Node overloading happens when the prediction function would compute an under-estimation of the load. In this case, less nodes than the necessary are recruited, causing a reduction in the QoS. To give the possibility to the operators to control the overloading, we define LF_{up} as an upper bound for the nodes load factor. LF_{up} is in the range [0,1], where 0 indicates that no upload bandwidth of nodes can be used, and 1 indicates that all the bandwidth is used for managing of VNs. By setting this parameter, operators can force the nodes to work under their capacity, as a node n will accommodate up to $LF_{up} * n_{cap}$ load. The LF_{up} parameter has to be carefully selected, as a high value can cause overloading, whereas a low value may cause resource over-provisioning. Ideally, LF_{up} should be able to cope with the error of the prediction function without affecting the economical cost of the infrastructure. An empirical evaluation of these factors, as well as the tuning of the LF_{up}, is presented is Section 5.

Due to the above considerations, we consider failures as the main cause of delayed updates. Delayed updates from failures depend on (i) the probability that a node fails during Δt and (ii) the time the infrastructure needs to recover from the failure, and (iii) the number of players accessing the node at the time of failure.

Assuming a model that describes failures of the nodes over time, we define $E[fail_n]$ as the expected probability for a node to fail during an arbitrary time step. When a node crashes, the backup node becomes the new node (see Section 2). The *failure time* (FT, in seconds) is the time that goes from the failure of a node to the moment the players have the information about the new node. Let us define T_f as the time-out needed for a backup node to notice the failure of the node. If a node does not communicate for T_f seconds with the backup node, it is considered failed. Also, let us define as $E[latency]$ as the expected latency between the backup node and the players. Then, $FT = T_f + E[latency]$. According to the definitions above we have:

$$DU(t) = (FT \times z) \sum_{n \in N(t)} (E[fail_n] \times n_{pl}(t)) \quad (3)$$

To let operators control the QoS, we then define the system-wide parameter α_{thres}. It represents the percentage of successful updates that must be kept by the platform. For instance, with $\alpha_{thres} = 0.99$, only 1% of updates can be delayed due to failures. Note that α_{thres} indirectly controls the assignment of the virtual nodes between user-provided and cloud nodes.

3.2 Problem statement

Our aim is to provide an assignment of the virtual nodes to the nodes that respects the bound defined by the operator to control the quality of service, while keeping the economical cost as low as possible. Hence, we define the problem of assigning virtual nodes to nodes as follows:

Problem Statement. *Find an assignment of virtual nodes to nodes to minimize $\beta(t)$ such that: $\alpha(t) \geq \alpha_{thres}$, and $n_{LF}(t) < LF_{up}$ for every $n \in N(t)$.*

4 Dynamic virtual node allocation

The task of the *manager* is to compute a virtual node assignment respecting the constraints defined in the problem statement in the previous section.

The work of the manager is divided into time intervals, which we refer to as *epochs*. Figure 3 shows the management of two consecutive epochs. During an epoch, the manager executes the following: (i) instantiates or releases on-demand nodes from the cloud, and migrates the virtual nodes according to the assignment plan done in the prior epoch (Section 4.3), and (ii) computes the new assignment for the next epoch (Section 4.2). The new assignment is computed with an heuristics based on the load prediction for the next Δt time units. Over time, the manager receives updates from the nodes about their load. These updates are not synchronized with the epochs. If an update arrives when the new

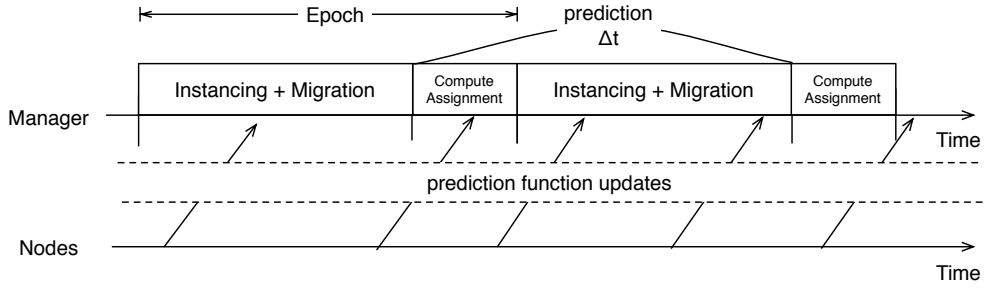

Fig. 3 Time management

assignment computation is already started, it will be considered in the next epoch.

The duration of an epoch (which we refer to as τ_{epoch}) must be tuned to accommodate the instantiation time provided by the cloud platform chosen, which normally is in the order of few minutes [33]. Due to the fact that we use an heuristics to compute the assignments, τ_{epoch} is largely occupied by the instancing+migration time. As a consequence, in the following we assume that $\Delta t \approx \tau_{epoch}$.

Algorithm 1: Server's load estimation

```
   Data: managerAddress, the IP of the manager
1  repeat
2      foreach v ∈ VN do
3          if |predictedLoad(L_v) − observedLoad| ≥ ξ_est
           then
4              L_v ← update(observedLoad);
5              msg ← add(v, L_v) ;
6      if msg.size ≠ 0 then
7          send(msg, managerAddress);
8  until true;
```

4.1 Load prediction

The manager computes the load of the virtual nodes by using a prediction mechanism for each of them. The manager stores, for a virtual node v, the data necessary to forecast the load of v at arbitrary time. We refer to this data as L_v.

For example, by considering a prediction mechanism based on a simple exponential smoothing:

$$v_{load}(t+1) = \alpha(\sum_{i=0}^{T} v_{load}(i)(1-\alpha)^{i-1}) \quad (4)$$

then L_v would be the historical observations of the load of v up to time t. Over time, the *manager* receives renewed load estimation functions from the nodes. Indeed, L_v is computed locally by each node, and then sent to the manager (see Algorithm 1). Periodically nodes check the error between the observed load value and the predicted value computed using L_v on the manager (line 3). If the error is larger than a predefined threshold ξ_{est}, then L_v is updated and sent to the manager (lines 4 and 7).

In our implementation we exploited an *exponential smoothing* function [21] to predict load trends. This model assured a good prediction power in spite of its simplicity. Nevertheless, the described approach in principle allows us to apply a wide range of statistical models for the load estimation, as for example autoregressive models for data prediction such as ARMA or ARIMA [37]. The choice of the model depends on the expected data fluctuations and the desired accuracy of the prediction ξ_{est}. ξ_{est} represents a reasonable error in the load estimation due to the choice of the estimation model. High accuracy estimation models predict the load trend for large times interval Δt ahead. On the other hand, these models require intensive computation and are not suitable for fast-pace applications like virtual environments. An in-depth analysis of different prediction mechanisms is left as future work.

4.2 Virtual Nodes Assignment

The virtual node assignment is computed by exploiting an heuristics and according to the predicted system state and the thresholds defined by the operator. For the sake of presentation, we divide the heuristics in two sub-tasks, *virtual node selection*, and *destination selection*.

Virtual Node Selection The aim of this task is to mark the virtual nodes to be migrated, adding them to vn_{pool}. Note that in this phase the manager works on an in-memory representation of the system, and that the actual migrations are executed once the virtual nodes assignment plan is defined.

The pseudo-code of this task is presented in Algorithm 2. Initially, the manager marks virtual nodes from overloaded nodes. Note that the manager compares the predicted load factors of the node at time $t+1$ against LF_{up} (line 2). The removal order of the virtual nodes considers the derivative of the virtual nodes' load trend. A virtual node with an high derivative would probably have a burst in the load soon, and migrating it may avoid overloading. Hence, the virtual nodes

Algorithm 2: Virtual Nodes Selection

input : LF_{up}, upper load factor threshold
input : P_{size}, the min amount of VNs to consider per epoch
input : α_{tresh}, QoS threshold
output: vn_{pool}, the list of virtual nodes to migrate

```
// Add VNs of overloaded nodes
1  foreach n ∈ N(t) do
2      while n_lf(t+1) > LF_up do
3          vn_pool ← maxDerivative(n_vn(t));
// Add VNs to control QoS
4  while α(t+1) ≥ α_tresh do
5      VP ← v ∈ VN : v is assigned to a user resource;
6      vn_pool ← maxDerivative (VP);
// Add backed up VN•
7  vn_pool ← vn_pool ∪ backUp();
// Removing VN from unused clouds
8  if size(vn_pool) < P_size then
9      VC ← v ∈ VN : v is assigned to a cloud resource;
10     Sort VC in ascending order according to nodes predicted
       load factor;
11     vn_pool ← (P_size − size(vn_pool)) VNs from VC;
// Anyway perturb the system
12 if size(vn_pool) < P_size then
13     vn_pool ← (P_size − size(vn_pool)) random VN;
```

Algorithm 3: Destination Selection

input : vn_{pool}, the list of virtual server to migrate
input : LF_{up}, upper load factor threshold
output: Actions, the list of migrations to execute

```
1  foreach v ∈ vn_pool do
2      Chosen = Null;
3      node_pool ← node_pool ∪ {n ∈ N(t): n_lf(t+1) < LF_up given
       (n ⊕ v)};
4      if node_pool is ∅ then
5          Chosen ← recruitNewCloud();
6      else
7          node_pool ← (n ∈ node_pool: α(t+1) > α_thres given
           (n ⊕ v));
8          if node_pool = ∅ then
9              Chosen ← recruitNewCloud();
10         else
11             Sort node_pool ascending according the cost;
12             Chosen ← node_pool.getFirst();
13     Actions ← migrate(v,Chosen);
14 executeActions();
15 releaseUnusedCloud();
```

with the highest derivative are marked for migration as first (line 3).

Afterwards, the manager marks virtual nodes for migration until $\alpha(t+1)$ is over the α_{thresh} defined by the operator (line 4). In this phase, only virtual nodes assigned to user-provided resources are considered (line 5). Moreover (line 7), the manager adds the virtual nodes currently managed by the back-up cloud nodes to vn_{pool} (see Section 2.6).

If, after these steps, the number of virtual nodes in vn_{pool} is less than P_{size} (line 8), additional virtual nodes are taken from cloud nodes with the lowest predicted load factor (line 11). This would lead to a removal of the unused cloud resources over time. If the size of the vn_{pool} is still lower than P_{size}, additional random virtual nodes are marked (line 13), to guarantee a constant level of perturbation to the system, useful to avoid being stuck in local optimal solutions.

Destination Selection The aim of this task is to assign the virtual nodes from vn_{pool} to nodes. The idea is to find, for each virtual node in vn_{pool}, a set of candidate nodes ($node_{pool}$), and then assigns the virtual node to the node candidate that minimizes the cost. The pseudo-code of this task is presented in Algorithm 3. Note that in the code we use the notation $n \oplus v$ to indicate a system where the node n would manage the virtual node v.

For each virtual node v in vn_{pool}, the manager first selects the node candidates such that, if assigned v, their predicted load factor would be less than LF_{up} (line 3). If no node satisfies this requirement, a new cloud node is recruited and v is assigned to it (line 5). Otherwise, the manager removes from $node_{pool}$ all the nodes that would decrease the QoS below α_{thres} (line 7). If no candidates remain in $node_{pool}$ after this further selection, a new cloud node is recruited. Otherwise, among the candidates left in $node_{pool}$, the manager selects the one minimizing the cost (line 12). Whenever all the virtual nodes are assigned, the manager performs all the migrations (see next section) and releases unused cloud nodes (lines 14 and 15).

4.3 Migration

At the start of the epoch, the manager executes the migrations that comes as output from the assignment computation of the previous epoch.

The migration procedure has been originally presented in our prior work [11]. We briefly explain it here with an example. Suppose that a virtual node V migrates from a source node A to a destination node B. The actions involved (presented in the sequential diagram of Figure 4) are the following:

1. The manager sends a reference to V and the address of recipient node B to node A.
2. A sends V to B, together with the list of users connected to V. In the transient time that is needed to complete the transfer, players still send entity update messages to A, which in turn forwards them to B. Note that in this transient period, entities may go out-of-sync and, as a consequence, players may perceive some visual inconsistencies.

3. Once received the message, node B notifies the clients that it has became the manager of V. From this point on, clients are able to modify the state of the entities included in V. However, the routing tables of the DHT have to be updated to assure correct routing resolutions.
4. To this end, V executes a *join* operation having B as target in order to update its references in the DHT. This operation updates the routing table of the node that are in the path from V to B, still leaving dangling references to A as the manager of V. To make consistent all references, the stabilization process of the DHT is executed.
5. Finally, a *leave* operation is executed by V on A in order to complete the process.

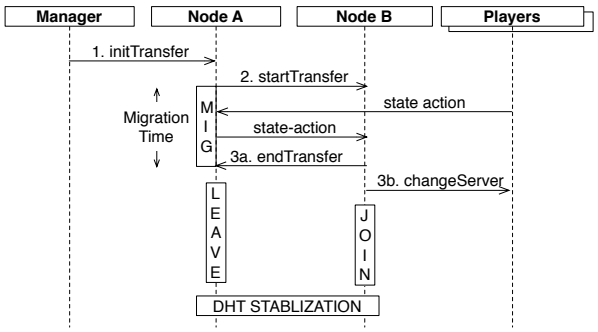

Fig. 4 Migration of a VN from the node A to node B

Since the objects inside the VN are not accessible to the clients during the transfer, this phase might affect users experience. To avoid this problem, it is important to limit the migration time of a virtual node, which largely depends on its size. A detailed tuning of the virtual nodes dimension is described in Section 5.2.

5 Experimental Results

The first part of this section presents the characteristics of the workloads used in our simulations. Then, we propose an empirical analysis to tune the dimension of the virtual servers and the maximum capacity threshold. Finally, we discuss the simulation results that consider cost and QoS varying different parameters, such as the number of players, the QoS threshold, and the churn level.

5.1 Workload Definition

A realistic simulation of the bandwidth load is central to properly evaluate a MMOG infrastructure[3]. The bandwidth load is sampled according to a discrete time step model. We define each step t to have a duration equal to Δt. For each step we compute the outgoing bandwidth requirement for the broadcasting of the entities of the players. The pricing model for bandwidth and renting is the one of Amazon EC2 [1]. In the following, we briefly describe the aspects considered when building the synthetic workload for our experimental setup; a more complete analysis is provided in [8])

Mobility models for players. Avatars move according to realistic mobility traces that have been computed exploiting the mobility model presented by Legtchenko et al. [32], which simulates avatars movement in Second Life [2]. We have presented this implementation, as well as a comparison with other mobility models, in [9]. In the model, players gather around a set of *hotspots*, which usually corresponds to towns, or in general to points of interest of the virtual world. Movements are driven by a finite-state automaton, whose transition probabilities are taken from the original paper [32].

The objects distribution. To place objects over the virtual environment, we use the same space characterization of hotspot areas used by the mobility model. A fraction of the objects is placed inside hotspot areas, so that their concentration follows a Zipfian distribution [42], with a peak in the hotspot center. The rest of the objects is randomly placed outside of hotspots. Figure 6 shows a snapshot of the placement of avatars and objects in the virtual environment.

The variation of the players number over time. Evaluating how the infrastructure adapts itself to variations in the number of players is an important task. In particular, since the load is in direct correspondence with the number of players, we are interested on how an increasing (and decreasing) load is managed by the infrastructure. We used two variation patterns of the players number. The first simulates the arrival and the leaving of a player according to a seasonal pattern, like the one described in Figure 5. In this pattern, the minimum value is set equal to the 10% of the maximum. The second pattern considers a stable number of users, i.e. their number does not change during the simulation. We generated two different workload from these patterns. In the rest of this section, we refer to the workload with the seasonal pattern as **W1**, and to the pattern with no variation as **W2**.

5.2 Tuning the Virtual Nodes Dimension

As we stated in Section 3, the *migration time* (MT) may affect the interactivity of the virtual environment. The more time a migration takes, the more the users perceive the virtual environment as “frozen”.

[3] We consider the load related to the management of the users. We do not take into account the bandwidth consumed for other tasks, like backup management, intra-server communications, and other services at application level (e.g. voice over IP).

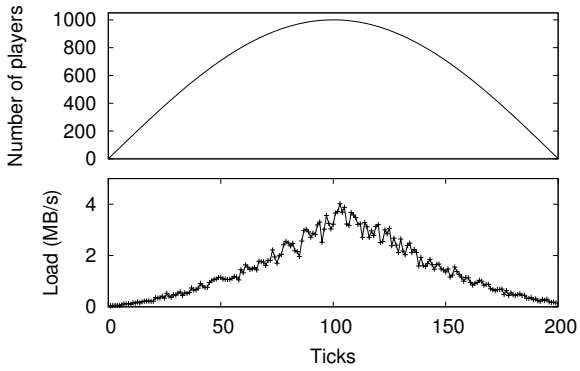

Fig. 5 Number of players over time (up) and correspondent load variation (down)

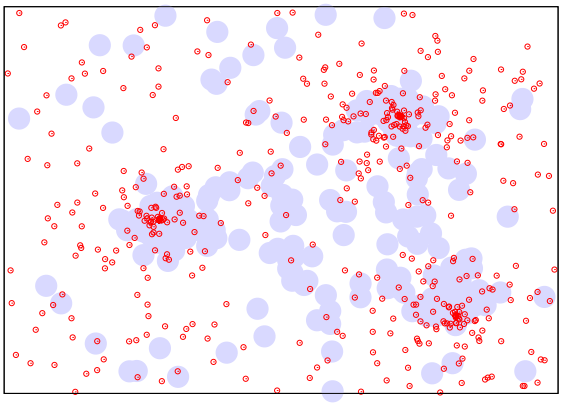

Fig. 6 Objects and avatars placement in the virtual environment

clients per entity per minute (hence, we consider a quite large timespan) in a simulation with synthetic generated avatars movements. The movements and the placement of the objects in the virtual environment were generated as described in Section 5.1. Figure 7 shows the histogram of the clients (in percentage) plotted in a log-log scale. The trend of the plot resembles a power law, i.e. a function of the form $y(x) = Kx^{-\alpha}$. By fitting the data, we derived $K = 0.5$ and $\alpha = 1.4$ (the corresponding function is also plotted in the figure). We exploited a number generator based on this function to generate the number of accessing clients for each virtual node, which is then multiplied for the length of the entry of the accessing clients list. An entry contains a UID (32 bits), a IP (32 bits) and a port (32 bits).

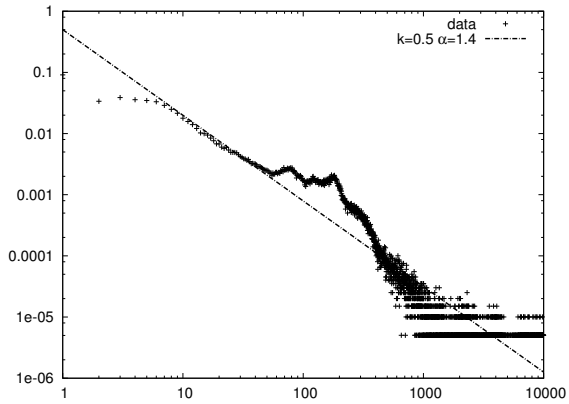

Fig. 7 Average clients per entity per minute plotted in log-log

The MT of a virtual node mostly depends on its size; indeed, it is important to tune this size to minimize this problem. The size of the virtual node depends on several factors: (i) the size of the routing table, (ii) the number of the clients accessing the virtual node, and (iii) the number of entities handled. In the following we enter in details of these three aspects.

The size of the routing table. The routing table of the virtual node contains the references to other DHT nodes, and depends on the particular DHT implementation chosen. In general, the routing table size is logarithmic with respect to the number of nodes participating in the DHT [46]. In our implementation we use Chord DHT [47] where each entry of the table is composed by a DHT-ID (160 bits) and a IP (32 bits). By considering a large DHT with 10K virtual nodes, the routing table contains 14 entries. Hence, the size of the routing table is 336 bytes.

The number of the clients accessing the virtual nodes. Any virtual node maintains a list of accessing clients. In order to estimate the number of connected clients per virtual node entity, we conducted an empirical analysis. We counted the

The number of entities handled. The content of an entity is composed by (i) a UID (32 bits), (ii) a DHT-ID (160 bits), (iii) a point representing the two-dimensional position in the virtual environment of the entity (32+32 bits), and (iv) a list of attributes, where to each entry name (32 bits) corresponds a respective value (64 bits). Let us assume that the dimension of the attribute list is arbitrarily fixed for all entities to 10 elements. We argue that this value is a good average estimate to contain enough information for a general MMOG. Summing up, each entity descriptor has a size of about 140 bytes.

We aim to determine the maximum size of a virtual node such that in the 95% of the cases the migration time (MT) takes less than the interactivity delay users may tolerate in response of their actions. This delay spans from few hundreds of milliseconds in fast-paced MMOGs up to two seconds in slow-paced MMOGs [15]. Here we stay in the middle, and consider the interactivity delay under 1 second as tolerable, which fits medium-paced game genre.

In order to find the maximum size of a virtual node we needed a MT model. The size of the virtual nodes were

generated by considering the aspects (size of routing table, accessing clients and entities handled) detailed above. To model MT we have exploited (with some minor modification) the model for TCP latency proposed by Cardwell et al. [7]. This TCP latency model requires Round Trip Times (RTTs) as input parameter. We model RTT delays according to the traces of the king dataset [24].

Based on the MT model we conducted experiments to study the influence of a size of virtual node on MT. Figure 8 shows that virtual nodes managing less than 15 entities have an MT less than one second with the 95th percentile. In the following, we use 15 as a maximum number of entities per virtual node.

More generally, this result may be used in two ways. Given a virtual environment with a predictable number of entities, it is possible to define the minimum number of virtual nodes to employ. On the other side, if a specific number of virtual nodes are needed, it is possible to know the maximum number of entities the system can support.

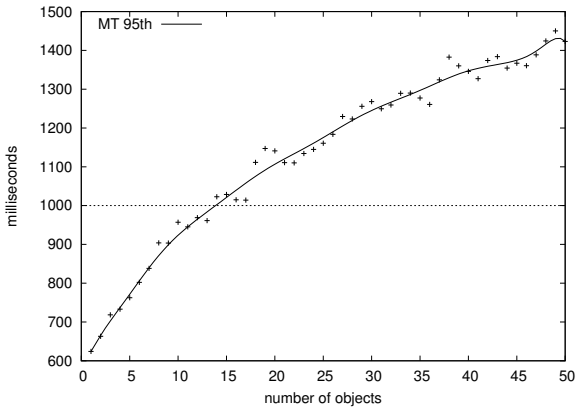

Fig. 8 95th percentile of MT with different amount of objects

5.3 Impact of a Virtual Node on the CPU

In order to understand whether a user-provided node can manage a VN, we deployed a prototype of our architecture on a real test-bed. For the experiment we considered a VN with 12 objects, accessed by 20 players concurrently and each player modifying the state of an object every 100ms (in total 200 messages/seconds arrive to the VN). With this setup, we measured the impact on the CPU of an average desktop workstation to be around the 6%. This suggests that a non-dedicated machine can manage a VN without compromising its performances.

5.4 Tuning the Capacity Threshold

In this section we discuss the evaluation of two parameters: (i) the maximum node capacity LF_{up} (see Section 3) and (ii) the error of the load prediction ξ_{est} (see Section 4). A proper tuning of these parameters is essential to avoid nodes overloading as well as resource under- and over-provisioning.

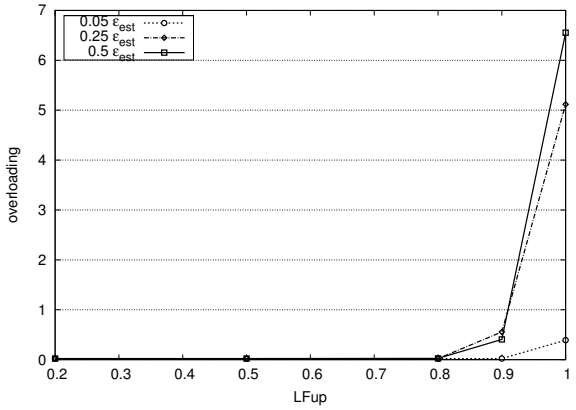

Fig. 9 Percentage of overloading over LF_up for different eps

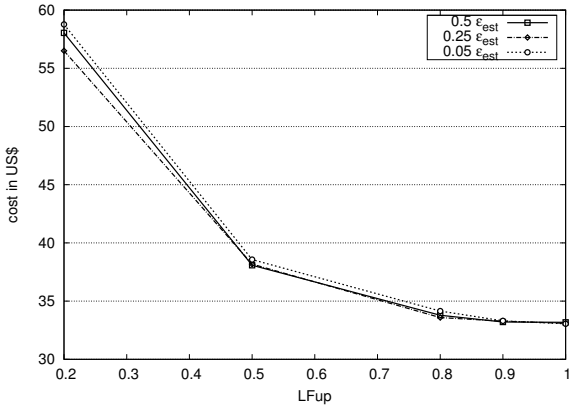

Fig. 10 Total simulation cost over LF_up with different eps

In our experiments we considered the value of LF_{up} to be the same for all the nodes. Figure 9 shows the percentage of overloading load for different LF_{up} values. As we can see from this figure, for a low $LFup$ the selection of epsilon is irrelevant. In particular, until $LF_{up} = 0.8$ the percentage of overloading remains around 0%. Nevertheless, setting $\xi_{est} = 0.05$ allows to tune $LF\_up = 0.9$ without nodes overloading and shows better shows in terms of resource utilization. By comparison, with $\xi_{est} = 0.25$ or higher, overloading starts at $LF_{up} = 0.8$. However, if we use all the capacity of the nodes ($LF\_up = 1$), the percentage of overloading grows up to 7%. This can be explained with the tendency of our prediction

mechanism to under-predict the load of the virtual nodes, causing the utilization of less resources then needed.

As we can see in Figure 10, the system cost decreases as LF_{up} increases. Nevertheless, for a high LF_{up} ($LF_{up} = 0.9$) the selection of the ξ_{est} is not relevant. The fact that with $\xi_{est} = 0.05$ we yield better results in terms of resource utilization, candidates $LF_{up} = 0.9$ and $\xi_{est} = 0.05$ as good values for the parameters. Note that these results hold for the chosen prediction mechanism, and should be recomputed if other prediction techniques are used.

5.5 Cost over the number of players

In this section we analyze the cost per minute over time and the total cost of a simulation with the seasonal (**W1**) and fixed (**W2**) access patterns, and with different number of players (from 1 up to 10 thousand). The QoS threshold, α_{thres}, is set to 0.95, $LF_{up} = 0.9$ and $\xi_{est} = 0.05$.

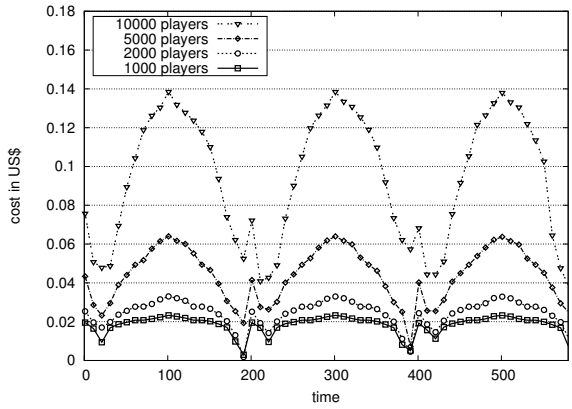

Fig. 11 Cost per minute over time considering workload 1, w1

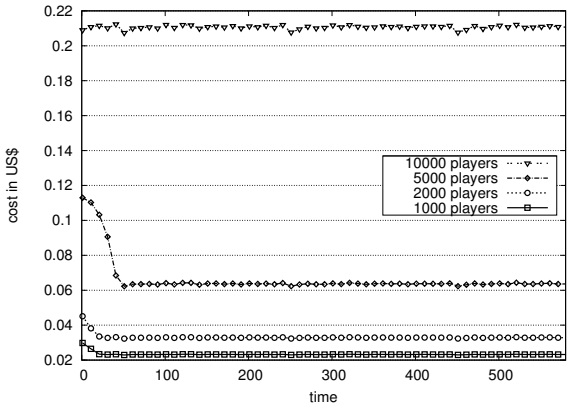

Fig. 12 Cost per minute over time considering workload 2, w2

Figures 11 and 12 show the cost per minute over time with **W1** and **W2** respectively. From the figures, it is evident that the cost largely depends on the number of players. The cost with **W1** has evident peaks and falls, and the cost per minute grows accordingly with the number of players. In this sense, the ability to exploit user-provided resources in situation of low load yields a clear advantage. On the contrary, the cost with **W2** is more stable, as the load is imposed only by the movements of the players in the virtual world and not by their access patterns. However, it is interesting to notice that the cost growing is not linear in terms of number of players and the cost for 10k users is almost 4 times higher than for 5k. This occurs since with a dramatic growing of load, user-provided resources became less able to serve virtual nodes.

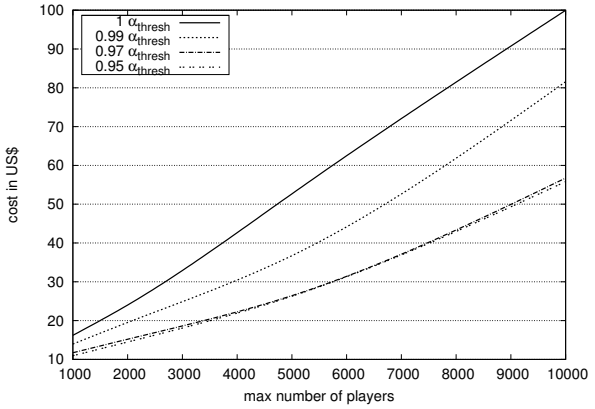

Fig. 13 Total cost with workload 1

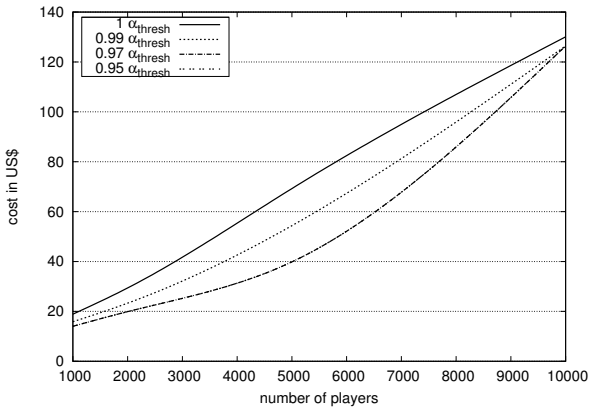

Fig. 14 Total cost with workload 2

Figures 13 and 14 depict the cost of a ten hours simulation for different number of players for **W1** and **W2** respectively. As it was expected, the cost with **W2** is higher due to the larger number of concurrent players participating to the virtual world. Moreover, as we can see from the figures, a lower QoS threshold allows to significantly reduce the total system cost. However, the difference of total cost

between $\alpha_{thres} = 1$ and $\alpha_{thres} = 0.99$ is 20% considering **W1**, whereas it drops to 1% when considering **W2**. Furthermore, the difference between $\alpha_{thres} = 1$ and $\alpha_{thres} = 0.95$ is around 60% in **W1**, whereas it is still 1% in **W2**. This suggests that with a fixed, maximum number of players, some virtual nodes are too heavily loaded to be managed by user-provided resources, even if those resources are available. This results in a larger utilization of cloud resources, which in turn increases the cost. Nevertheless, the system with **W2** shows good results in case of medium size load (around 5k users) and allows to reduce up to 60% of the cost between $\alpha_{thres} = 1$ and $\alpha_{thres} = 0.95$.

Figures 13 and 14 also introduce another interesting observation. The cost of the two workloads when α_{thres} is 0.95 and 0.97 are basically the same. This suggests that a further reduction in the α_{thres} would have no impact on the cost of the platforms. This occurs because no virtual nodes can be assigned to user-provided resources, due to their limited bandwidth capabilities. The correspondence of results with α_{thres} equal to 0.95 and 0.97 is also present in the next section.

5.6 Qos and Cost Trade-off

One of the strength of our approach is to provide MMOG operators the ability to set the desired level of QoS of the platform. The operator chooses the *QoS threshold*, α_{thresh}, which affects the assignment of virtual nodes to nodes. In general, the higher the threshold, the less virtual nodes are assigned to user-provided resources. As a consequence, this threshold indirectly controls the operational cost of the platform.

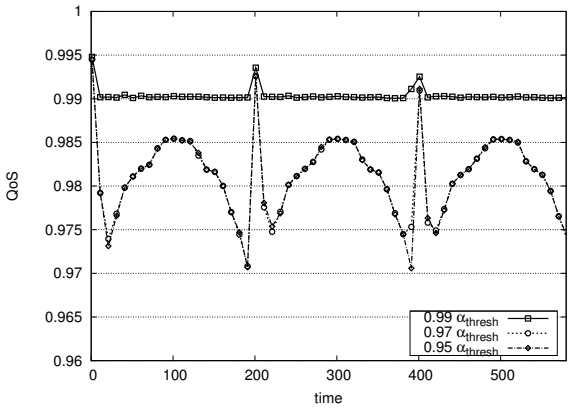

Fig. 15 QoS over time with different QoS thresholds

To evaluate the trade-off between cost and QoS we considered the workload **W1**, $\xi_{est} = 0.05$, $LF_{up} = 0.9$ and 5k players. Figure 15 shows how the architecture maintains the QoS in the system above the specified QoS level. At first, as we can see from the figure, the architecture successfully maintains a level of QoS above the threshold. However, the system QoS never reaches the threshold of 0.95. This is because all the virtual nodes that can be assigned to user-provided resources have been already assigned. One more point is when the level of the QoS raises up, regardless of the threshold. These increments occur when cloud resources are exploited to supply the low number of user-provided resources. For instance, around the 200th and 400th ticks according to the seasonal pattern there are no available user-provided resources to support the system (see Figure 5).

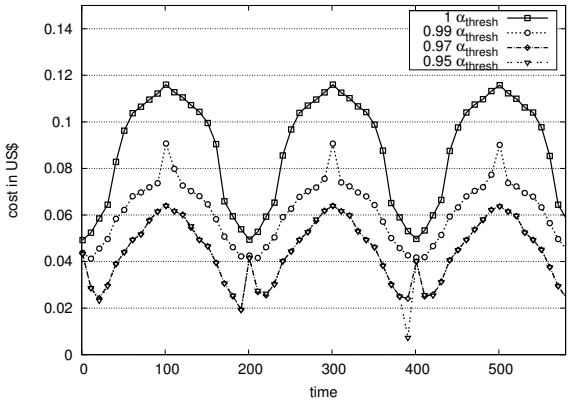

Fig. 16 Cost per minute with different QoS thresholds

Figure 16 shows the cost per minute for different QoS thresholds. In the situations of peak load and lack of user-provided resources the utilization of cloud increases, in fact increasing the cost as well. This suggests that on-demand resources are essential in order to support the MMOG.

5.7 Churn analysis

By employing user-provided resources, our system is exposed to the effects of the players' churn, i.e. the process of players leaving and joining the MMOG over time. We considered two aspects of churn. The first aspect is related to the seasonal access pattern of players, which considers the churn over long periods of time. This aspect is modelled by considering a workload with a variable total number of players over time (see Section 5.1). The second aspect regards the churn during short periods of time. This kind of churn does not affect the "big picture" of the seasonal access pattern of players, but instead generates an interchange of players during a small fraction of time. We modelled this second aspect by assigning to the probability of players interchange the values 0.05, 0.1 and 0.5.

Figure 17 shows the results in terms of QoS. In the experiments we used $\alpha_{thresh} = 0.95$. The figure suggests that the level of churn directly affects the minimum level of QoS reached. In case of high churn level, the amount of user-

provided resources is reduced to maintain an acceptable level of QoS. For instance, with a churn level of 0.05 the QoS stays around 0.99, whereas with a churn level of 0.5 the QoS drops to the threshold parameter 0.95. This is confirmed in Figure 18, which measures the cost at the same conditions. As we can see with a churn level of 0.5 the cost is significantly higher than in the other cases.

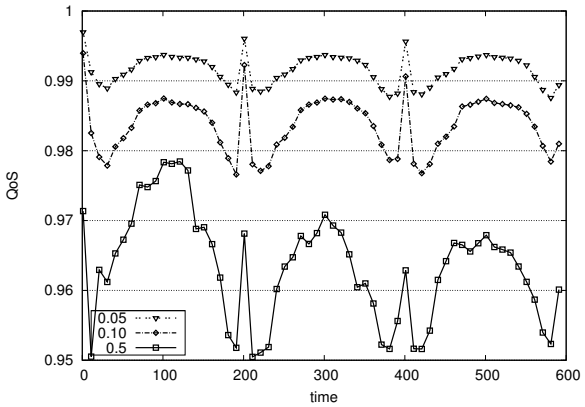

Fig. 17 QoS over time with different churn levels

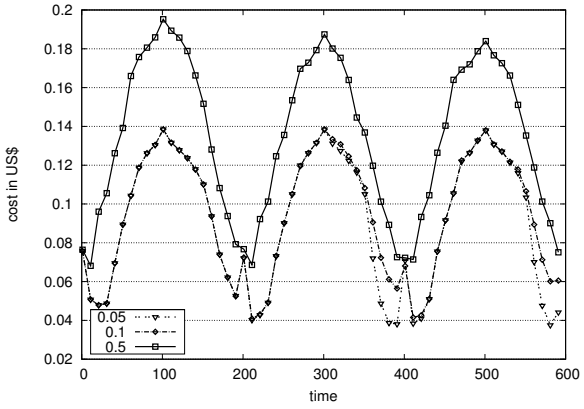

Fig. 18 Cost over time with different churn levels

6 Related work

In this section we frame our work in two macro-areas: integration of P2P and Cloud Computing (Section 6.1) and hybrid architectures for MMOGs (Section 6.2). In Section 6.3 we collect and compare approaches that, to the best of our knowledge, are more relevant with respect to the provisioning of resources in Cloud-operated MMOGs.

6.1 Integration of P2P and Cloud Computing

Combined cloud and P2P infrastructures recently gained attention from the research community. These infrastructures aim to resolve the issues typical of pure P2P solutions.

A common problem is the asymmetry in the bandwidth capability of user-provided resources, where the download bandwidth is usually much higher than upload. In content distribution and information dissemination, a widely used approach to enable the upload of the content from user- provided resources is to use the so called *helpers*, i.e. additional cloud nodes supporting the delivery of content [36,48,49, 38].

A MMOG infrastructure can be seen as a content delivery system, where the content is the state of the virtual environment. In this sense, the addition of cloud to support user-provided servers in the management of the MMOG, is comparable to adding helpers. However, unlike classical content delivering, the state of the MMOG is modified by the players themselves, so servers shall manage not only the diffusion but also the consistency and correctness of the information.

An essential property in hybrid P2P-cloud architectures is *self-configuration*: components and protocols are autonomically configured according to specific target goals (such as reliability and availability). In dynamic contexts, self-configuration is often supported by forecasting the resource utilization [52] and orchestrating the leasing and releasing of pay-per-use resources. In our previous work [30] we exploited a resource prediction mechanism to self -tune the amount of replicas on top of an hybrid P2P and cloud system. We leveraged this experience to proactively assign the objects of the MMOG to the nodes of the infrastructure.

6.2 Hybrid MMOG architectures

Last decade has seen the rising of several P2P-based architectures targeting large-scale on-line games. Among them, the hybrid MMOG architectures focused on the combination of centralized resources with P2P paradigms to support MMOGs. To the best of our knowledge, this work is the first proposing the integration of P2P and cloud computing as a solution for MMOGs.

Hybrid MMOG architectures aim to exploit and combine user-provided resources (peers) and centralized resources (servers) in a seamless infrastructure. A central issue for hybrid MMOGs is to define a partitioning of the virtual environment, so that it is possible to assign specific tasks to peers. A widely-used method is *spatial partitioning*, i.e. the virtual environment is divided into *regions* or *cells*, whose dimension can be either fixed or variable. These regions are in turn assigned to a peer or a server, that becomes the *manager* of the entities in such regions. Kim et al. [31] consider a virtual environment partitioned in square regions, initially

managed by a central server. The first peer with enough computational and bandwidth capabilities to enter a cell becomes the cell manager. Afterwards, a fixed number of peers entering the same cell act as backup managers in order to increase failure robustness. Similarly, Barri et al. [3] proposes an hybrid system including a central server and a pool of peers. The central server runs the MMOG and, as soon as it reaches the maximum of its capacity, it delegates part of the load to the peers.

Other approaches employ a *functional* partitioning of the MMOG, where only subset of functions are delegated to peers. For instance, Chen and Muntz [14] proposes a functional partition of the MMOGs tasks. Central servers are responsible for user authentication and game persistence while manage only regions characterized by high-density user interactions, whereas peers support low-density interaction regions. Jardine and Zappala [28] provide a distinction between *positional* and *state-changing* actions. They propose an hybrid architecture where peers manage positional actions, that are more frequent and prone to be maintained locally. Central servers handle state-changing actions, that are not transitory and require a larger amount computational power. By comparison, our general architecture exploits a similar characterization of actions, while the functional partitioning is done at the level of components rather than at resource type. In other words, we employ two different components that respectively manage positional and state-changing actions.

With respect to the state of the art, we stay somewhere in the middle. On one hand, we consider some functionalities, like authentication, to be handled by centralized and full controllable servers. On the other hand, other functionalities may be mapped to both central servers and peers. This requires a more flexible, dynamic strategy for region distribution, to allow for a fine-grained management of the resources by the MMOG operator. Resources control is very important for our approach, because the seamless combination of cloud computing and P2P requires to keep under control the cost and effectively deal with the implicit uncertainty related to peers.

6.3 Resource provisioning in Cloud Computing

With the advent of Cloud Computing there is the tendency to decouple resource management of applications and services from the underlying infrastructure. This requires mechanisms and strategies for adapting the provisioning of resources to the variable workload of applications and to meet the given QoS. A number of works in this field are focused on general purpose architectures for adaptive resource provisioning [12,27,18], whereas other considered more specific challenges, such as cost optimization [22,13] and energy saving [29]. Along with general-purpose works, application-specific proposals such as scientific [17] and data streaming [43] have been proposed.

In this context, MMOG architectures are not exceptions. In the last years, several works have been proposed with the specific aim of resource provision in on-demand infrastructures according to MMOGs workloads and QoS. In the following we describe and compare the works that, at best of our knowledge, show most analogies with our paper. We identified five main features that characterize provisioning for MMOG services (see Table 2):

- *underlying infrastructure*;
- *feedback model*, which differentiates between responding to monitoring or employing some type of prediction to allow a pro-active behavior;
- *provisioning principles*, describes the principles that drive the provisioning;
- *migration*, whether the impact of migration is considered when orchestrating provisioning;
- *cost analysis*, whether a cost analysis of the proposed solution is performed.

D. Ta et al. [50] proposes a set of algorithms to optimize resources in a MMOG platform, while maintaining the network latency as low as possible. The assignment of clients to servers is done purely based on the latency, as they assume servers to have unlimited computing and bandwidth capabilities. However, the authors considered a fixed-pool of resources, therefore they do not provide mechanisms to acquire and releasing resources. As a consequence, migration is not considered in the affection of the QoS and the paper does not provide an analysis of cost related with the management of the platform. Similarly to us, Marzolla et al [34] present an adaptive Cloud resource provisioning for MMOG. Their system replies accordingly to the level of the QoS (i.e. latency between servers and clients) adding or releasing resources. However, their system is not pro-active as they do not consider load prediction when computing the amount of necessary resources. In a series of papers [39, 40], Nae et at. present a comprehensive architecture for the management and on-demand provisioning of resources for a MMOG platform. Their work takes into account the impact of migration on the QoS, employs a prediction system and provides cost analysis. The main difference with our work is in the fact that the authors do not consider directly the cost of the platform as a mean to drive the provisioning. One more difference is in the integration of user-provided resources into the system.

corollary

7 Conclusion

The combination of cloud and P2P computing is currently an hot topic in various applications, including large scale on-

-	Underlying infrastructure	Feedback model	Provisioning principles	Migration	Cost analysis
D. Ta et al. [50]	server clusters	N/A	network latency	no	no
Marzolla et al. [34]	cloud	monitoring	response time	no	no
Nae et al. [39,40]	cloud	prediction	resource type and network latency	yes	yes
Our work	cloud, user resources	prediction	server overloading, cost	yes	yes

Table 2 Comparison among approaches employing adaptive resource provisioning

line games. The embedded flexibility of these architectures is a valuable asset for operators, that can choose which kind of resources to use. In this direction, this paper proposes a hybrid architecture merging the different characteristics of P2P and cloud nodes to offer a valid support for large scale on-line games. An efficient and effective provisioning of resources and mapping of load are mandatory to realize an architecture that scales in economical cost and quality of service to large communities of users.

Our heuristics tackles these issues by strategically migrating the load and recruiting new resources. We also provide operators with the ability to control the behavior of the platform. By selecting a desired QoS level, operators can control the amount of cloud and user-provided resources to exploit. This allows performing aggressive strategies with the aim of reducing the cost. We demonstrated that trading a negligible amount of QoS cuts the cost of the platform up to 60%. Furthermore, the approach is proved to be robust in case of high system load and extremely high level of network churn.

Due to its flexibility and generality, our work opens additional research paths that can be explored. For instance, an analysis with different load prediction mechanisms would adapt our techniques to other kind of workloads. Another interesting improvement would be to adapt our technique to the possibility to use different cloud providers at the same time, in order to increase economical effectiveness. Furthermore, the investigation of NAT traversal solutions might increase applicability of our approach in realistic network conditions.

References

1. Amazon elastic compute cloud (Amazon EC2). `http://aws.amazon.com/ec2/`, [Online; accessed Aug-2012]
2. Second life website. Http://secondlife.com/, [Online; accessed Jan-2013]
3. Barri, I., Giné, F., Roig, C.: A Scalable Hybrid P2P System for MMOFPS. 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing pp. 341–347 (2010)
4. Barri, I., Gine, F., Roig, C.: A Hybrid P2P System to Support MMORPG Playability. 2011 IEEE International Conference on High Performance Computing and Communications pp. 569–574 (2011)
5. Bharambe, A., Douceur, J., Lorch, J., Moscibroda, T., Pang, J., Seshan, S., Zhuang, X.: Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. ACM SIGCOMM Computer Communication Review **38**(4), 389–400 (2008)
6. Buyya, R., Yeo, C., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems **25**(6), 599–616 (2009)
7. Cardwell, N., Savage, S., Anderson, T.: Modeling tcp latency. In: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol. 3, pp. 1742–1751. IEEE (2000)
8. Carlini, E.: Combining peer-to-peer and cloud computing for large scale on-line games. Ph.D. thesis, IMT Institute for Advanced Studies Lucca (2012)
9. Carlini, E., Coppola, M., Ricci, L.: Evaluating compass routing based aoi-cast by mogs mobility models. In: Proceedings of the 4th International Conference on Simulation Tools and Techniques, pp. 328–335. ICST (2011)
10. Carlini, E., Coppola, M., Ricci, L.: Reducing Server Load in MMOG via P2P Gossip. In: Proceedings of the 11th Annual Workshop on Network and Systems Support for Games (NetGames) (2012)
11. Carlini, E., Ricci, L., Coppola, M.: Flexible load distribution for hybrid distributed virtual environments. Future Generation Computer Systems (2012)
12. Casalicchio, E., Silvestri, L.: Architectures for autonomic service management in cloud-based systems. In: Computers and Communications (ISCC), 2011 IEEE Symposium on, pp. 161–166. IEEE (2011)
13. Chaisiri, S., Kaewpuang, R., Lee, B.S., Niyato, D.: Cost minimization for provisioning virtual servers in amazon elastic compute cloud. In: Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on, pp. 85–95. IEEE (2011)
14. Chen, A., Muntz, R.: Peer clustering: a hybrid approach to distributed virtual environments. In: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games, p. 11. ACM (2006)
15. Claypool, M., Claypool, K.: Latency and player actions in online games. Communications of the ACM **49**(11), 40–45 (2006)
16. Cronin, E., Kurc, A., Filstrup, B., Jamin, S.: An efficient synchronization mechanism for mirrored game architectures. Multimedia Tools and Applications **23**(1), 7–30 (2004)
17. Deelman, E., Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: the montage example. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, p. 50. IEEE Press (2008)
18. Espadas, J., Molina, A., Jiménez, G., Molina, M., Ramírez, R., Concha, D.: A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures. Future Generation Computer Systems **29**(1), 273–286 (2013)
19. Ford, B., Srisuresh, P., Kegel, D.: Peer-to-peer communication across network address translators. In: USENIX Annual Technical Conference, General Track, pp. 179–192. USENIX (2005)
20. Frey, D., Royan, J., Piegay, R., Kermarrec, A., Anceaume, E., Le Fessant, F.: Solipsis: A decentralized architecture for virtual environments. Proceeding of 1st International Workshop on Massively Multiuser Virtual Environments (MMVE'08) pp. 29–33 (2008)
21. Gardner, E.: Exponential smoothing: The state of the art—part ii. International Journal of Forecasting **22**(4), 637–666 (2006)

22. Ghanbari, H., Simmons, B., Litoiu, M., Iszlai, G.: Feedback-based optimization of a private cloud. Future Generation Computer Systems **28**(1), 104–111 (2012)
23. Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I.: Load balancing in dynamic structured P2P systems. In: INFOCOM 2004. Twenty-third Conference of the IEEE Computer and Communications Societies, pp. 2253–2262. IEEE (2004)
24. Gummadi, K., Saroiu, S., Gribble, S.: King: Estimating latency between arbitrary internet end hosts. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment, pp. 5–18. ACM (2002)
25. Hu, S.: Spatial Publish Subscribe. Proc. of IEEE Virtual Reality (IEEE VR) workshop, Massively Multiuser Virtual Environment (MMVE,Äô09) (2009)
26. Hu, S., Chen, J., Chen, T.: VON: a scalable peer-to-peer network for virtual environments. Network, IEEE **20**(4), 22–31 (2006)
27. Huber, N., Brosig, F., Kounev, S.: Model-based self-adaptive resource allocation in virtualized environments. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 90–99. ACM (2011)
28. Jardine, J., Zappala, D.: A hybrid architecture for massively multiplayer online games. In: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, p. 60. ACM (2008)
29. Kantarci, B., Mouftah, H.T.: Optimal reconfiguration of the cloud network for maximum energy savings. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), pp. 835–840. IEEE Computer Society (2012)
30. Kavalionak, H., Montresor, A.: P2P and cloud: A marriage of convenience for replica management. In: Proc. of IWSOS'12, pp. 60–71. Springer (2012)
31. Kim, K., Yeom, I., Lee, J.: HYMS: A hybrid mmog server architecture. IEICE Transactions on Information and Systems **E87**, 2706–2713 (2004)
32. Legtchenko, S.: Blue Banana: resilience to avatar mobility in distributed MMOGs. Networks pp. 171–180 (2010)
33. Markatchev, N., Curry, R., Kiddle, C., Mirtchovski, A., Simmonds, R., Tan, T.: A cloud-based interactive application service. In: e-Science, 2009. e-Science'09. Fifth IEEE International Conference on, pp. 102–109. IEEE (2009)
34. Marzolla, M., Ferretti, S., D'angelo, G.: Dynamic resource provisioning for cloud-based gaming infrastructures. Computers in Entertainment (CIE) **10**(3), 4 (2012)
35. Mauve, M., Vogel, J., Hilt, V., Effelsberg, W.: Local-lag and timewarp: Providing consistency for replicated continuous applications. IEEE Transactions on Multimedia **6**(1), 47–57 (2004)
36. Michiardi, P., Carra, D., Albanese, F., Bestavros, A.: Peer-assisted content distribution on a budget. Computer Networks **56**(7), 2038–2048 (2012)
37. Montgomery, D., Jennings, C., Kulahci, M.: Introduction to time series analysis and forecasting, vol. 526. Wiley (2011)
38. Montresor, A., Abeni, L.: Cloudy weather for P2P, with a chance of gossip. In: Proc. of P2P'11, pp. 250–259. IEEE (2011)
39. Nae, V., Iosup, A., Podlipnig, S., Prodan, R., Epema, D., Fahringer, T.: Efficient management of data center resources for Massively Multiplayer Online Games. 2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis pp. 1–12 (2008)
40. Nae, V., Prodan, R., Fahringer, T.: Cost-efficient hosting and load balancing of massively multiplayer online games. In: Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on, pp. 9–16. IEEE (2010)
41. Nae, V., Prodan, R., Iosup, A., Fahringer, T.: A new business model for massively multiplayer online games. In: Proceeding of the second joint WOSP/SIPEW international conference on Performance engineering, pp. 271–282. ACM (2011)
42. Newman, M.: Power laws, pareto distributions and zipf's law. Contemporary physics **46**(5), 323–351 (2005)
43. Payberah, A.H., Kavalionak, H., Kumaresan, V., Montresor, A., Haridi, S.: Clive: Cloud-assisted p2p live streaming. In: Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on, pp. 79–90. IEEE (2012)
44. Roverso, R., El-Ansary, S., Haridi, S.: Natcracker: Nat combinations matter. In: ICCCN, pp. 1–7. IEEE (2009)
45. Saito, Y., Shapiro, M.: Optimistic replication. ACM Computing Surveys (CSUR) **37**(1), 42–81 (2005)
46. Steinmetz, R., Wehrle, K.: Peer-to-Peer Systems and Applications, vol. 3485. Lecture Notes in Computer Science (2005)
47. Stoica, I., Morris, R., Liben-nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. IEEE/ACM Transactions on Networking (TON) **11**(1), 17–32 (2003)
48. Sweha, R., Ishakian, V., Bestavros, A.: Angels in the cloud: A peer-assisted bulk-synchronous content distribution service. In: Proc. of CLOUD'11, pp. 97–104. IEEE (2011)
49. Sweha, R., Ishakian, V., Bestavros, A.: AngelCast: cloud-based peer-assisted live streaming using optimized multi-tree construction. In: Proc. of MMsys'12, pp. 191–202. ACM (2012)
50. Ta, D.N.B., Nguyen, T., Zhou, S., Tang, X., Cai, W., Ayani, R.: Interactivity-constrained server provisioning in large-scale distributed virtual environments. Parallel and Distributed Systems, IEEE Transactions on **23**(2), 304–312 (2012)
51. Wu, J.: The World of MMORPG: a Tale of Two Regions (2010). URL `strategyanalytics.com`
52. Wu, Y., Wu, C., Li, B., Qiu, X., Lau, F.: Cloudmedia: When cloud on demand meets video on demand. In: Proc. of ICDCS'11, pp. 268–277. IEEE (2011)