On demand Resource Allocation Middleware for Massively Multiplayer Online Games

André Pessoa Negrão, Miguel Adaixo, Luís Veiga and Paulo Ferreira
Distributed Systems Group, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa
{andre.pessoa,miguel.adaixo,pjpf,luis.veiga}@tecnico.ulisboa.pt

***Abstract*—Traditionally, commercial MMOGs are deployed on large privately owned server clusters with hundreds of computing devices linked through high bandwidth connections. In this scenario, the dynamic and unpredictable workload variability of MMOGs frequently leads to resources being under/over used with negative impact on *playability* and/or cost-effectiveness. In our research work, we see cloud computing as a fundamental approach to mitigate the problem of inefficient resource provisioning, due to its inherent elasticity properties. Thus, we propose a *cloud-aware* middleware for MMOGs, in which virtual machines obtained from cloud providers are added and removed from the system according to load changes observed at runtime. In this paper, we report on our first steps towards such an infrastructure.**

I. Introduction

Supporting large scale and highly interactive applications such as Massively Multiplayer Online Games (MMOGs) requires an infrastructure capable of processing and disseminating a large amount of data with high performance and constant availability. To meet these requirements, commercial MMOG companies deploy large and expensive private server clusters provisioned with high bandwidth connections and powerful machines; for example, World of Warcraft3 uses more than 10000 servers [1], while EverQuest4 employs more than 1500 servers over 13 data centers [2]. Within these infrastructures, servers are statically assigned to manage a portion of the game's virtual world. To cope with workload variability, the allocation of resources to each partition is based on worst case scenario predictions on the number of players that are expected to be located in that partition. This strategy results in an over–provisioned environment in which a large number of resources are idle for most of the time. In fact, a study shows that in Second Life about 30% of the servers are empty all the time, while only 1% of the servers are continuously accessed [3].

Despite its adoption among commercial games, over-provisioning is a very inefficient strategy that results in unnecessary expenses. Nevertheless, the larger game companies are willing to pay this price, since over-provisioning i) is easy to implement and manage and ii) ensures that the game delivers the interactivity levels demanded by the players. However, over-provisioning is a conservative *fix* that is not economically sustainable in the long term. In addition, it is not an option for small companies, which are prevented from trying to make their way into the MMOG environment.

*This work was partially supported by national funds through FCT – Fundação para a Ciência e Tecnologia, under projects PEst-OE/EEI/LA0021/2013 and PTDC/EIA-EIA/113993/2009.

We claim that the inherent elasticity properties of the cloud computing paradigm [4] make it an ideal tool to provide cost–efficient resource provisioning for MMOGs. Thus, in this paper we present CloudDReAM (**D**ynamic **Re**source **A**llocation **M**iddleware), our first iteration towards a cloud-based dynamic resource provisioning middleware for MMOGs. CloudDReAM automatically allocates resources from a cloud computing provider whenever the current resources prove insufficient to handle the load imposed on the system. In addition, it disposes of previously acquired resources when they are no longer necessary. This way, CloudDReAM allows game companies to deploy their own private server clusters without over–provisioning them, and resort to the services of a cloud provider when their own resources are not sufficient to handle the load. As a result, resources are used more efficiently and the costs of supporting the game are reduced, benefiting every stakeholder involved.

This paper is structured as follows. Sect. II discusses related work. Sects. III and IV describe the architecture and implementation of CloudDReAM. Sect. V presents the experimental results obtained so far. Finally, Sect. VI concludes the paper.

II. Related Work

Several authors have already identified the advantages of using the cloud to improve the cost–effectiveness of MMOGs. Nae *et al.* proposed a solution that uses neural networks to predict the load that will affect the game in the near future [5]. Based on this prediction and the current load, the system executes a load balancing algorithm that decides which actions to take (e.g., add/remove a server) [6]. When adding a new server, however, the system executes a global load balancing algorithm that includes every active server in the process, a solution that is not scalable. In our work, we aim at a more localized and lightweight solution.

Glinka *et al.* [7] proposed a cloud infrastructure based on the *Real-Time Framework* [8]. In their solution, load balancing only considers replication-based policies (e.g., add more replicas or migrate users between replicas). Replication, however, has limited scalability, as it requires each server to have global knowledge of the system. As the system grows, the overhead of processing every object becomes a bottleneck and the performance of the system decays.

Marzolla *et al.* proposed another cloud based solution in which servers are either gateways, application servers or databases [9]. They then use a Queueing Network model to

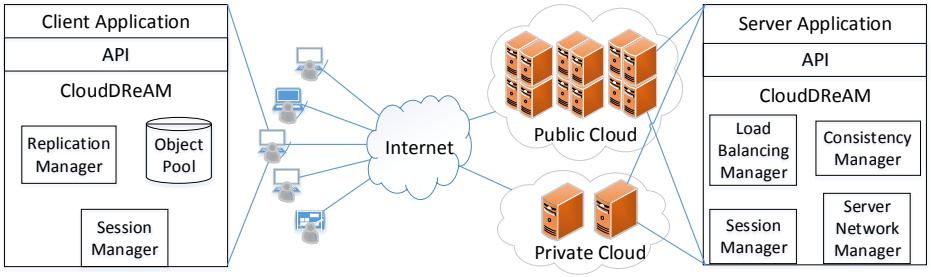

Figure 1. High level architecture of CloudDReAM.

choose from different resource distribution alternatives, with the goal of minimizing the number of resources assigned to each tier. Their work assumes that every server in the same tier has the same load. As a result, their only load balancing option is to add a new server when the system (as a whole) is overloaded. This approach is also not cost–effective, since it does not attempt to redistribute load among the existing servers before adding a new one.

Najaran *et al.* [10] propose renting servers from the cloud and organize them in a P2P network. However, the authors do not discuss how the system performs load balancing. In addition, the system employs multicast–based update dissemination, assigning one multicast address to each object, which is not scalable.

III. CloudDReAM

The CloudDReAM middleware (Fig. 1) follows a Client/Server architecture in which servers, consisting of Virtual Machines (VMs), are acquired from a public cloud provider (e.g., Amazon EC2[1]). VMs can also be acquired from a private cloud managed by the developers of the MMOG, if available. We distinguish both as *public* and *private* VMs, respectively. We assume no particular type of initial setup, both regarding the number and type (public or private) of VMs present. At runtime, however, we prioritize acquiring private VMs whenever possible.

At the client side, the middleware abstracts the application from such aspects as data replication and communication with the server. It receives the events generated by the local user and updates the application with information received from the servers. It also maintains a replica pool consisting of the objects that are of interest to the player, managed according to Interest Management techniques [11]. For this purpose, we use the Vector-field Consistency model [12].

At the server side, CloudDReAM is responsible for abstracting the server application from the underlying organization of the system. In addition, each CloudDReAM server manages the consistency state of a subset of clients and manages load balancing. The server comprises three core components: 1) the Server Network Manager, which deals with server–to–server communication; 2) the Consistency Manager, responsible for consistency and interest management; and 3) the Load Balancing Manager, which takes care of load balancing aspects.

[1]http://aws.amazon.com/ec2

Our approach to workload distribution is to partition the virtual world into dynamically sized rectangular partitions. Each partition is assigned to a server which becomes responsible for the game objects (including players' avatars) located in it. Despite this partitioning, players can freely interact with each other and avatars can move around the virtual world, transparently switching between partitions.

Each server possesses only a partial view of the network, i.e, it knows only a subset of the total number of servers of the network. The partial view of a server always contains the servers responsible for adjacent partitions (neighbor servers). In addition, it may also contain a number of other servers, added to the partial view for load balancing and consistency management.

One of the servers of the system takes on the special role of *cloud manager*, becoming responsible for coordinating load balancing. Servers contact the cloud manager when they identify a relevant load event. The cloud manager analyzes the load status of the server and decides on the best option to solve the load event.

Each CloudDReAM server continuously monitors its load status in order to identify relevant load situations. The load status of a server is composed of one or more load metrics defined by the developers of the game; these metrics can be, for example, the CPU or bandwidth usage. For each load metric defined, corresponding *underloaded* and *overloaded* threshold values are established; when a threshold value is reached, load balancing is issued. When a server identifies a threshold violation, it asks the cloud manager to initiate the load balancing process. Depending on the load event detected (under or overload), the server executes one of two algorithms, as we explain next.

1) Server Overloaded: When a VM is overloaded, it needs to share its workload with another VM. Since renting a VM from a public cloud encompasses additional costs, CloudDReAM prioritizes sharing load with private resources (which have a fixed maintenance cost, as they have already been paid for). When that is not possible, it proceeds by obtaining a public VM. The algorithm runs as follows.

a) Share load with new private VM: The cloud manager first verifies if it is possible to launch a new private VM. The role of the new VM depends on the type of VM in which the overloaded server is being executed (*overloaded VM*):

- If the overloaded VM is public, the cloud manager

verifies if it can substitute it with the private VM. This verification aims at ensuring that after the substitution the private VM does not become overloaded. If that is the case, the cloud manager proceeds with the substitution, terminating the previously overloaded public VM.

- If it is not safe to substitute the public overloaded VM or the overloaded VM is private, the cloud manager splits the overloaded VM's partition in two, assigning one of its halves to the newly created private VM and the other to the previously overloaded VM.

b) Share load with active VM: If no private VM is available, the cloud manager analyzes the load status of the neighbors of the overloaded VM. If any of these is capable of receiving the extra workload associated with managing one half of the overloaded server's partition, the cloud manager assigns that half to it. The advantage of sharing load with neighbors is that it limits the number of servers that need to communicate during the standard operation of the system. In addition, it minimizes the migration overhead, since neighbors already share some information about each others' state.

c) Share load with new public VM: If the previous options are not possible, the cloud manager tries to acquire a public VM as follows.

- If the overloaded server runs on a public VM, the cloud manager first analyzes if it possible to substitute it with the newly acquired VM. If so, it proceeds with the workload migration and can, then, dispose of the overloaded VM; once more, the goal is to minimize the costs of renting public resources. Otherwise, the partition is split between the two VMs.
- If the overloaded VM is private, the cloud manager proceeds in the opposite manner: it first tries to split the partition between the private and the new VM; if that is not possible, it migrates the state (replacing the private VM with the public one).

2) Server underloaded: Having an underloaded server means the game is incurring in unnecessary expenses. As such, CloudDReAM tries to improve workload distribution by, if possible, removing an underloaded VM from the system. The algorithm executed in an underload situation runs as follows.

a) Public VM underloaded: When the underloaded VM is public, CloudDReAM first verifies if the partitioned managed by the overloaded server can be merged with any of the neighbor partitions. If so, the two partitions are merged and assigned to the neighbor server. Otherwise, CloudDReAM verifies if it is possible to migrate the workload of the VM to a private VM (either an active or a new one). If none is available, the VM is not removed.

Even when the system detects that a VM should be removed, the removal process is not necessarily carried away immediately. The reason for this is that, typically, VMs are acquired for a pre-established amount of time and are paid for in advance. As a result, removing the VM before the expiration of the rental period has no advantage in terms of cost reduction. In addition, maintaining the VM until the expiration date allows the system to respond to future overload situations in a more efficient and stable way.

b) Private VM underloaded: If the underloaded VM is private, there is no advantage in removing it, since the costs of executing it are fixed. As such, in order to make a more efficient use of the available resources, we try to maximize the use of the private, self–owned resources by searching for a public VM to substitute.

IV. Implementation

We implemented a prototype of CloudDReAM in C# using Microsoft's *.Net* platform. The implementation of the middleware follows the design described in Sect. III and depicted in Fig. 1. CloudDReAM provides an API for applications to interact with the middleware. The API provides functions for the application, both at the client and the server side, to read and update the objects in the pool, as well as adding and removing objects, among others.

To get insight from the perspective of a developer using CloudDReAM, we extended the multiplayer game *Cube 2: Sauerbraten*[1] over our middleware. Our choice for Sauerbraten was due to the availability and good documentation of the source code. Despite being a First Person Shooter, it allowed us to experiment our middleware with a real game. Our extension to Sauerbraten required us to modify only 5 source files (plus a few header files for includes). In total, we wrote less than 300 lines of code, most of which were located on a few methods. In all, the footprint of CloudDReAM in the implementation of the game is significantly small, specially when compared to the game's full implementation, which comprises more than 240K lines of code.

V. Evaluation

Being in the preliminary stages of evaluation, we conducted a simple test case to analyze our middleware and identify improvements for future work. The test case starts with two servers to which 200 simulated players are connected. Then, 100 more players are added to the simulation every minute, up to a total of 600 players. Players are evenly distributed across the simulated virtual world and move according to the random waypoint mobility model. We compare the results obtained by CloudDReAM with two static infrastructure scenarios, one with 2 fixed servers and the other with 8. Our simulations were executed on a four node cluster of Intel Core2 Quad Q6600 2.40GHz machines with 8GB of RAM, connected through a 100Mbps LAN.

In our experiments we considered two main load metrics, CPU usage and frame rate. The frame rate is a measure of the throughput of the game that broadly corresponds to the number of times (*frames*) per second that a server is able to provide players with information about game events. We consider as overload threshold for this metric 5 or less frames per second. As for the CPU metric, we consider (based on measurements we made) 60% CPU usage as our overload threshold.

Figure 2 shows the CPU usage results. Labels *Avg_2s* and *Avg_8s* show the average CPU usage for the 2 and 8 servers

[1] http://sauerbraten.org/

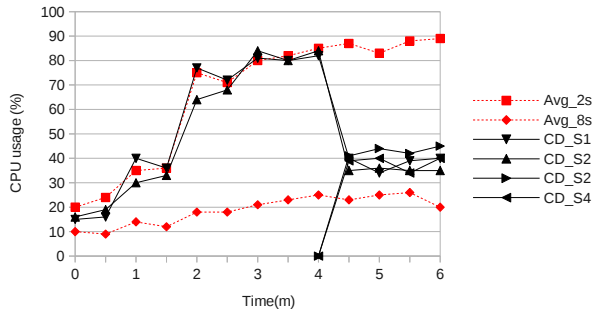

Figure 2. Server load

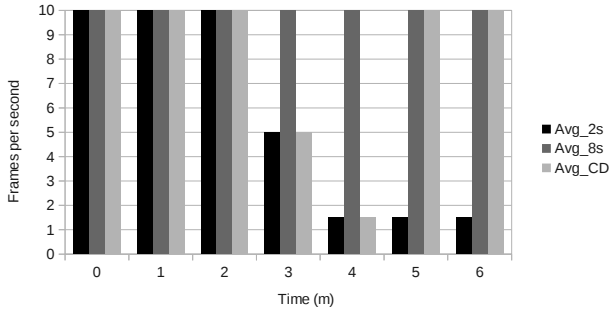

Figure 3. Frame rate

static scenarios, respectively. Labels *CD_S1* to *CD_S4* show the CPU load of the CloudDReAM servers. Figure 3 shows the average frame rates obtained by the static and CloudDReAM scenarios. For both figures, when averages are shown it is because the values of the different servers are identical.

The first thing that is clear from the figures is the inefficiency of static provisioning. Specifically, the 8 servers scenario (labeled *Avg_8s* in the figures) is an example of overprovisioning, in which the server load averages 20% of CPU usage and never surpasses the 25% mark. On the other hand, the 2 servers scenario exemplifies an underprovisioning situation. In this case, the performance of the system starts decaying 5 minutes into the simulation (the time when the number of players amounts to 500), as shown by the drop to 5 frames per second (*fps*). When the number of players increases to 600, the frame rate drops to 1.5 fps, which would result in a performance decay observable by the players.

CloudDReAM, on the other hand, is able to make a more efficiently use of resources by adapting to load at runtime. This adaptation is initiated at the 2 minute mark, the time when the two active servers detect that their CPU load is above the overload threshold. At this moment, the servers contact the Cloud Manager, which identifies the necessity of launching two new VMs, each to share the load with one of the active VMs. Due to the delays associated with booting the VMs, the new servers take between 2 and 3 minutes to become fully operational. During this delay, the performance of the system is affected, dropping to 5 fps at minute 3 and then to 1.5 fps at minute 4. When the VMs finally become operational, the performance is restored to the desired levels and the frame rate increases back to 10 fps.

The main conclusion we extract from this evaluation is the need to add a load prediction component to CloudDReAM. With this component, CloudDReAM would be able to anticipate impending overload situations and launch VMs sooner, preventing the performance decays that currently occur while VMs are starting up. In addition, this component should be complemented with fallback mechanisms that are able to minimize the performance loss when predictions fail. We intend to address these issues in future work.

VI. Conclusions

In this paper we proposed CloudDReAM, our first step towards a cost–effective resource provisioning middleware for MMOGs. CloudDReAM monitors the load on the servers and issues load balancing when the load measured reaches pre–established thresholds. If necessary, CloudDReAM obtains new resources from a cloud provider or disposes of previously acquired VMs when it determines they are no longer needed. Our solution allows game infrastructures to be deployed conservatively (w.r.t. resource quantity) and scale continuously according to runtime load, allowing game managers to pay for infrastructure according to demand. As such, the costs of maintaining the game are reduced and the active resources are used more efficiently, opens the gaming environment to new companies that would, otherwise, be forced to incur in a high and risky initial investment.

References

[1] S. Gorlatch, F. Glinka, and A. Ploss, "Towards a scalable real-time cyberinfrastructure for online computer games," in *ICPADS 2009*, 2009, pp. 722 –727.

[2] D. Kushner, "Engineering everquest," *IEEE Spectrum*, vol. 42, no. 7, pp. 34–39, Jul. 2005.

[3] M. Varvello, S. Ferrari, E. Biersack, and C. Diot, "Exploring second life," *IEEE/ACM Trans. Netw.*, vol. 19, no. 1, pp. 80–91, Feb. 2011.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[5] V. Nae, A. Iosup, and R. Prodan, "Dynamic resource provisioning in massively multiplayer online games," *IEEE Trans. on Par. and Dist. Syst.*, vol. 22, no. 3, pp. 380 –395, March 2011.

[6] V. Nae, R. Prodan, and T. Fahringer, "Cost-efficient hosting and load balancing of massively multiplayer online games," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 2010, pp. 9–16.

[7] S. Gorlatch, D. Meilaender, A. Ploss, and F. Glinka, "Towards bringing real-time online applications on clouds," in *Computing, Networking and Communications (ICNC), 2012 International Conference on*, 2012, pp. 57–61.

[8] F. Glinka, A. Ploß, J. Müller-lden, and S. Gorlatch, "Rtf: a real-time framework for developing scalable multiplayer online games," in *ACM SIGGCOM NetGames '07*. ACM, 2007, pp. 81–86.

[9] M. Marzolla, S. Ferretti, and G. D'Angelo, "Dynamic resource provisioning for cloud-based gaming infrastructures," *Comput. Entertain.*, vol. 10, no. 3, pp. 4:1–4:20, Dec. 2012.

[10] M. T. Najaran and C. Krasic, "Scaling online games with adaptive interest management in the cloud," in *ACM SIGGCOMM NetGames '10*. IEEE Press, 2010, pp. 9:1–9:6.

[11] K. L. Morse, "Interest management in large-scale distributed simulations," University of California, Irvine, Department of Information and Computer Science, Technical Report ICS-TR-96-27, Jul. 1996.

[12] L. Veiga, A. Negrão, N. Santos, and P. Ferreira, "Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency," *J. Internet Services and Applications*, vol. 1, no. 2, pp. 95–115, 2010.