Cloud DReAM - Dynamic resource allocation management for large-scale MMOGS

Miguel António Moreira de Sousa Adaixo

[1] Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal
[2] Distributed Systems Group, INESC-ID, Portugal
miguel.adaixo@ist.utl.pt

Abstract. In the last few years massive multiplayer online games (MMOG) have been gaining an ever increasing number of adepts. This type of systems raise serious challenges such as ensuring scalability of the system while maintaining game playability (i.e. the overall game performance in order to make the game an enjoyable experience).
Most common implementations of these systems rely on a client-server architecture, using different types of approaches to distribute the load among the various computers. We believe that the cloud computing paradigm is an interesting approach given its elasticity properties. In this work we propose a cloud computing platform in order to solve the major issues that a MMOG has to deal with, focusing on minimizing the resource waste while ensuring playability. This means that the cloud servers must react immediately to any change of load resulting from a variable number of clients playing the game.

Keywords: Cloud Computing, Elasticity, Resource Management, Multiplayer Online Games

1 Introduction

1.1 Motivation

A Massively multiplayer online game (MMOG) is a genre of game in which a very large number of players interact with one another within the game's virtual world. This type of interaction uses the internet as a support medium, and requires a persistent game world in which the player's actions take place.
This type of system is a clear example of a distributed system. Taking this into account, most of the common problems with distributed systems apply in this type of game, and in some cases can be even harder to solve than on regular distributed systems. The two main distributed systems concerns that are dealt with in this case are scalability and usability of the application.
Typically such games run on top of a client-server architecture, with many clients connecting to one server where the game world is hosted. Clients want to play the game as smoothly as possible no matter how big the number of players currently connected to the server. This can pose a serious challenge to the server,

which has to process many clients at the same time, and respond to them in a timely manner. It becomes clear that this process can be very heavy on resource consumption on peak hours, when most players are connected.
Thinking about the costs of all this operation, it is easy to realise that it will be very high, especially on those hours where the number of clients connected is lower. During these periods of lower resource demand, it is not justified to have resources being wasted on potential clients that just will not connect within that time periods. In order to solve this problem we want to develop a system that allows for this resource management to be done automatically and with minimal impact on the game performance, preventing the waste of resources when they are not needed.

1.2 Objectives

The main objective of this work is to create a distributed system that has the capability to dynamically adjust the resources it uses to support MMOGs. This means increasing or decreasing the number of resources used, in order to support an ever changing number of players. This adaptation performed by the system, should be done in an automatic and timely fashion to serve the needs of the users. It also has to be done in such a way that it allows for the usability of the game to be kept unchanged as much as possible. It is also necessary for the cost of the resource usage to be kept to the minimum value possible while maintaining the service quality on par with what is currently done. Considering what has been said, our requirements for this system are scalability and elasticity of resources that are used to support the game. We also need to maintain the performance and usability of the game in the eyes of the users.

1.3 Difficulties/Problems

There are a series of difficulties related to what we are trying to achieve. One of them is what we call the threshold problem: when is it time to add or remove resources? In order to minimize resource usage and the corresponding cost, this has to be taken into account. Furthermore, it is important to find what is the best metric to use for such threshold. The aspects that should be considered when trying to solve this problem are the number of players connected, the load of the servers, the communications taking place between servers among some other metrics.
The next problem we have to deal with is game consistency. When we add resources, in this case more server machines to support the increasing number of players, we have to ensure that the game world remains consistent in both the new machines and the older ones. This is also relevant when we remove machines that are no longer needed. It is mandatory to ensure that any information present on the disconnecting machines is kept so that players are not affected by any loss of data.
The performance of the game is another difficulty that we face in this system.

It is necessary that the users do not experience long negative impact due to what is happening in the background while they play. What this means is that gameplay should be kept smooth independently of the addition or removal of machines that might be taking place.
Finally, it is very important to consider scalability, since it is a main concern in this kind of systems. To begin with, there are two common architectures being used for MMOG, which are the peer-to-peer (commonly known as P2P), and the client-server architecture [2]. P2P has some advantages when compared to client-server. This advantages are related to transmission latency and the absence of a single point of failure. It is however hard to prevent cheating and to manage the game state using this architecture. For this reason most solutions rely on the classic client-server architecture which solves the disadvantages of P2P at the expense of the number of servers and bandwidth consumption. The typical client-server architecture that is in use nowadays is not the most scalable approach and game companies often have to find difficult to manage solutions to make it scale. Some might argue that P2P would present itself as a more scalable and effective solution. Despite its many positive aspects, P2P lacks in the security, game state persistence and availability aspects, which are critical for an MMOG. We aim at a solution that has enough elasticity to deal with the fluctuation in resource usage, adapting its resources automatically.

2 Cloud DReAM

In this work we decided to use the cloud computing approach. Cloud computing has a series of characteristics which greatly contributed to our decision. This characteristics are in line with our requirements of scalability, elasticity of resources, performance and usability. The main characteristic that weighted on our decision was the elasticity aspect of clouds. Furthermore, this is also an approach that has not been widely explored for MMOG, in contrast with the other two possible approaches that we have considered, and that have had extensive research on this subject.

2.1 System Overview

On a very high level view our system is represented in Fig. 1, where a number of game clients connects to the game servers that are being hosted by our cloud platform. Each of the servers present in the cloud is running a previously loaded image of the game server. Clients connect to one of the servers according to the cloud's load balancing policies. The resources (in this case the servers) being used to support the game are managed dynamically. The game can start with one single server, and due to the interactions happening inside the game world, it might be necessary to add more resources to support the increasing demand on the servers. The opposite is also true, the number of resources being used can be decreased, if the current demand does not justify their presence. This

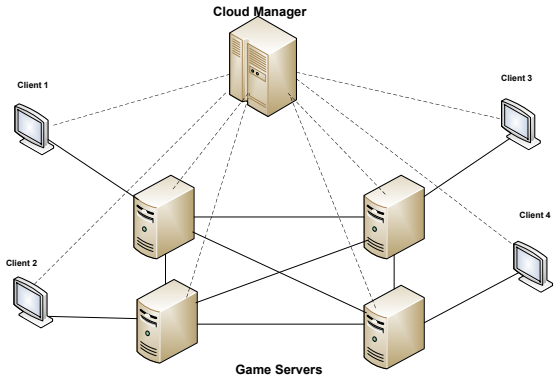

Fig. 1. Representation of the interaction between the 3 components of the system

management is performed automatically by the system and the player is not aware of what is happening in the background.
The Cloud DReAM middleware system acts between the client and the cloud server. It is split between the clients, the servers and the cloud manager. On the client side, it is responsible for dealing with the client status updates having into account its area of interest (AOI). The client's status is kept by the middleware as well as the status of other players that might be relevant. To achieve this, Cloud DReAM implements an interest management algorithm known as Vector Field Consistency (VFC) [8]. This information is then used to decided what is to be presented to the player. The communication between client and the middleware platform is possible through an API.
Servers are the main enforcers of the system consistency. The VFC interest management algorithm is controlled by the servers. Each server applies the VFC technique for the clients currently connected to it. On the server side, it is also necessary to take into account the load balancing issues. Cloud DReAM is responsible for dealing with this aspect. To fulfil this task, it is important to consider how the players are distributed among the servers. Cloud DReAM uses an algorithm which divides the game world into different areas that are managed by different servers. The consequence of this is that players will connect to the server that is responsible for the game area where their avatar is currently located. The map division is done using an algorithm [5] that splits the map into regular rectangle shaped areas. Based on this information, the system takes the appropriate measures. These measures are the migration of clients between servers based on their position inside the game world.
The servers are capable of keeping some state locally, but since they are volatile machines that can be removed when they are not needed, it is necessary to guarantee that any state they had is kept. Cloud DReAM can use the tools provided by the cloud platform in order to coordinate the state transfer between the vir-

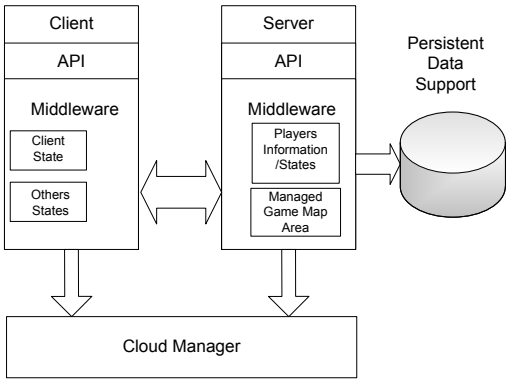

Fig. 2. System components architectural view

tual machines and, if needed, the persistent storage of the game information.
The third component of the system is the cloud manager. This component is the direct responsible for the operations performed on the cloud platform. The decisions related with scaling and load balancing are managed by this component. In order to perform its function, the cloud manager receives information from clients and servers. On the servers case, it receives from them their CPU usage % every thirty seconds. Based on this information it decides if the system need to scale up or to scale down. The cloud manager also provides the clients and the servers with information they need to operate. Fig. 2 illustrates an architectural view of the various components of the system, and their relations.
All the actions of our system are performed in the background, and need to be as efficient as possible, in order to maintain the performance and enjoyment of the game. Any effort to scale and optimize game resources is not useful if the playability is compromised by that process.

3 Implementation

The system prototype was developed using the Cube2 [1] game. This game is written in C++. We have created our system using C# and .NET and it interacts with the original cube game as a library that can be invoked through an API.
We had to perform a conversion of the original game, to make it work in a multi server environment. The original solution only used one server to manage each game sessions, which would not be useful for our solution. We have also added the cloud manager component to the system. This component is responsible for some of the functions that were previously responsibility of the server. An example of one of these functions is the distribution of unique player identifications

[1] http://sauerbraten.org/

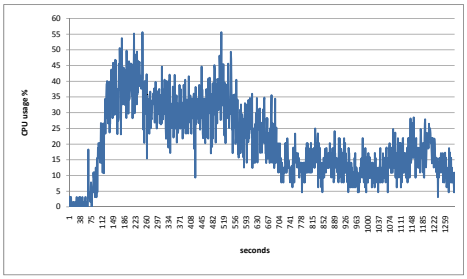

Fig. 3. CPU Usage for server 1(Cloud DReAM second case)

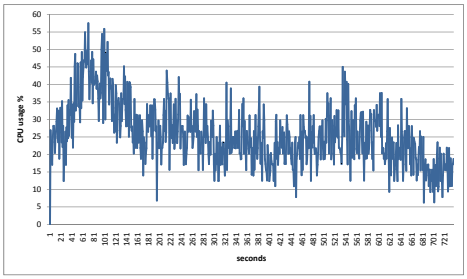

Fig. 4. CPU Usage for server 2 (Cloud DReAM second case)

to every client that connects to the game.
The communication between all the components is performed using the remoting functionalities of the .NET platform. Depending on the type of communication that is performed, the protocol used can be either UDP or TCP. For periodic object updates, UDP is being used, since there is no real need to guarantee that every update is received. It is a lighter protocol in terms of the communication overhead. For events we use TCP since these are not periodic and are important to be guaranteed to arrive. An example of an event is a player shooting another player in the game.
The cloud platform being used is the Eucalyptus version 3.1 [1]. The cloud manager communicates with the cloud platform through SSH protocol. This communication is used to invoke specific Eucalyptus commands that allow the launch and termination of new server instances in the cloud.

4 Evaluation

4.1 Tests Description

In order to evaluate the system, we conceived three main testing scenarios. To obtain a baseline scenario, we performed a test using a single server running the original version of the game. This shows us the expected behaviour of a single server, and the type of load to expect for a given number of players. We also performed another test, using four statically allocated servers, that were active during the entire duration of the test. The purpose of this test is to observe what is the load on these four servers, with the same number of players used for the first test. We expect to understand if the resources used (in this case four servers)

[1] http://www.eucalyptus.com/

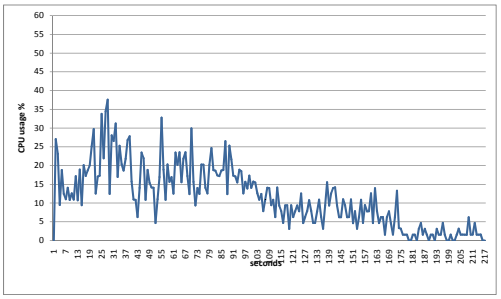

Fig. 5. CPU Usage for server 3 (Cloud DReAM second case)

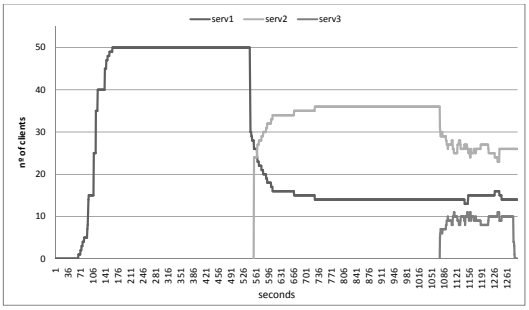

Fig. 6. Clients connected to each server per second

are adequate to the needs of the game. Finally we performed a test using Cloud DReAM. On this scenario, we started with a single server, and connected the same number of players as in the previous test cases. The system then adapted the resources used according to the needs of the game.
On these tests we used 50 players controlled by artificial intelligence, running for approximately 20 minutes. In Figs. 3 through 8 we present the results from the last experience described.
We also performed usability tests with real players. For these tests we asked players to play the game without our system, and with our system in order to understand if there were noticeable differences.

4.2 Cloud DReAM test

On this test one server is connected and players start to join the game. After a while, the server considers itself overloaded and asks the cloud manager to scale. When the second server is ready and connects, a large portion of clients immediately migrates to that server. This sudden migration happens because of the game world division. A large portion of the players is now located in the area of the second server, and thus is migrated to that server. This increase in the server load causes the second server to consider itself overloaded and requests the cloud manager for help. This generates the launch of a third server. Fig. 3 through Fig. 5 show the loads for the different servers and in Fig. 6 we have the players distribution across servers. They illustrate what we have just described. We can see an increase in the CPU load on server one around 186 seconds, that triggers the scaling operation. The chart also shows that another load spike occurs on server one right after the scaling operation. This does not trigger a new scale because the cloud manager knows there is a pending scale

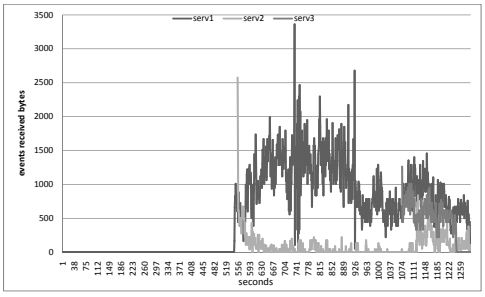

Fig. 7. Events received for the 3 servers (Cloud DReAM second case)

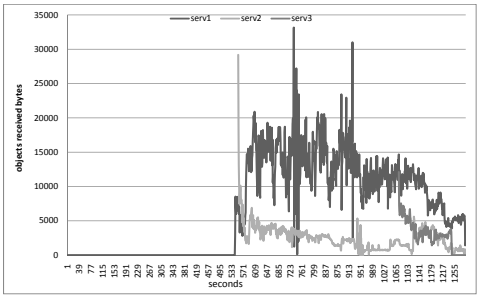

Fig. 8. Objects received in the 3 servers (Cloud DReAM second case)

request from server 1. After server 2 finishes the connection process, we observe the increase in its CPU load around 81 seconds (with relation to server 2 time scale) and the consequent scale request that generates the launch of server 3. Finally, nearing the end of the test, we can see that server 3 has a CPU load below 5% starting from 175 seconds (with relation to server 3 time scale). This causes this instance to be terminated. The machine was terminated because its load was low, and the cloud manager decided that it would not cause additional stress on the remaining two machines.

In Fig. 7 and Fig. 8 we show the network values for this scenario. The charts start with no communication until the second server connects. It is clear that the network communication is one of the causes for the extra load we see after a split, and this experience confirms it. We can compare this case with a different experience, where we had the same conditions, but the network values were lower. On that case the system only required two servers to maintain a stable game. On the current experience, the system required 3 server to deal with the extra load it was experiencing, but eventually stabilized and decided to terminate the third instance. When we compare the Cloud DReAM scenario with the static test performed, we can draw some conclusions. The first thing that we notice is that the scenario that is using the Cloud DReAM system uses less servers to support the same number of clients. This supports our claim that there is a waste of resources in the static approach. Furthermore this shows that our system can in fact improve the usage of resources, by dynamically deciding when to scale or reduce the number of resources that are required. Another observation that we make is that we can further optimize the communication between servers as well as the load balancing algorithms being used. We see by our tests that the load balancing algorithm based on areas of the map is not ideal for the Cube 2 game, and can generate weird situations. The improvement of the load balancing algorithm will allow for an even better resource usage, that

will be closer to the ideal case of a perfect split of the load.
On the usability tests the players found some differences on both versions. The differences found are related with some glitches that happen due to the division of players across many servers. Bugs such as shot animation not showing properly or score tables being reset after a migration. While this bugs are important, they can be considered minor issues. The main focus of our concerns were the impact of player migrations between servers. On this subject, users did not notice that the migration happened, which meets our goal. The game performance was not affected either, since no user mentioned this aspect.

5 Related work

There are some other already existent approaches that try to solve the same challenges as our system. We will describe them briefly here and discuss their advantages and disadvantages.
Project Darkstar [9] is described as a platform that proposes to solve all the hard issues related with the development of MMOG, one of them being the addition or subtraction of resources as needed, and leaves to the programmers only the task of creating a fun game. Darkstar is designed to exploit the multi-threaded, multicore chips that exist today. It also aims to scale to a large group of machines while giving the programmer the illusion of being developing in a single-threaded, single-machine environment. The project has since lost its funding due to restructuring in the company where it was being developed, and was not concluded.
There is a lot of research on the subject of using P2P for MMOG. As it was previously stated, P2P seems like a very natural solution to scalability for a MMOG environment. In order to have a P2P system working for an MMOG type of game, there are some problems [3] that are viewed as essential to be solved in order to have a successful system. Some of these problems have been addressed in some systems, but to our knowledge have not been solved by a single system.
In one of these P2P approaches [4], the authors claim that it is possible to take advantage of P2P by creating an overlay network that uses a series of components to achieve its goals. The first component is Pastry [7], a DHT-based Overlay Network, which provides the organization and structure that will allow the system to tolerate network failures as well as a high number of connecting peers. For object management, an extension to Pastry called Past is being used, which allows the persistence of objects. The last component is Scribe, another extension of Pastry that is meant to deal with multicast, which is used to disseminate the game events to all the peers. This approach has been tested with an MMOG that is not time critical. The authors claimed their approach suffices in that case, but they have not conducted tests on a more typical MMOG. They are in fact time-critical, and latency is a serious issue.
A different solution based on P2P [6] which uses the advantages of this type of

architecture in order to achieve the resource efficiency that we are looking for. It does however need to have some special nodes or trusted peers that control the security aspect of the game, which makes the system not able to fully exploit a P2P architecture. Another proposed approach was a MMOG support system [1], that presented a solution for most of the MMOG problems. However, scalability is a bit neglected and relieved to a second plane of focus. The test setting for this approach is also somewhat limited which does not make it clear how the system would behave on a larger scale.

6 Conclusions

In this work we have analysed the research that has been done in the MMOG area, specially in what concerns the resource usage and its optimization. We have discussed several possible approaches to the resource management in MMO environments and analysed each approach weaknesses and strengths. We have proposed our own approach that relies on cloud computing as its basis. We implemented and evaluated the results obtained. Our results are encouraging, they show that there is potential in using the cloud computing approach to optimize the resource usage of MMOG's. We have managed to achieve our requirements, even if only partially in some cases. There are still optimizations that can be performed, namely on the communication between servers, and on the algorithms used for load balancing and scaling.

References

1. Assiotis, M., Tzanov, V.: A distributed architecture for mmorpg. Netgames (October 2006)
2. Duong, T.N.B., Zhou, S.: A dynamic load sharing algorithm for massively multiplayer online games. IEEE (2003)
3. Fan, L., Trinder, P., Taylor, H.: Design issues for peer-to-peer massively multiplayer online games. School of Mathematical and Computer Sciences , Heriot-Watt University
4. Hampel, T., Bopp, T., Hinn, R.: A peer-to-peer architecture for massive multiplayer online games. Netgames (October 2006)
5. Negrao, A.F.P.: Vfc large-scale: consistency of replicated data in large scale networks. Instituto Superior Técnico (September 2009)
6. Rieche, S., Wehrle, K., Fouquet, M., Niedermayer, H., Petrak, L., Carle, G.: Peer-to-peer based infrastructure support for massively multiplayer online games. RWTH Aachen University / University of Tubigen
7. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. Proc. of The 18th IFIP/ACM International Conference on Distributed Systems Platforms (November 2001)
8. Veiga, L., Negrao, A., Santos, N., Ferreira, P.: Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency. INESC-ID, Lisboa Portugal
9. Waldo, J.: Scaling in games and virtual worlds. Communications of the ACM 51 (August 2008)