Università
della
Svizzera
italiana

Faculty
of Informatics

Research Prospectus

July 5, 2011

A fault tolerant multi-server system for MMOGs

Carlos Eduardo B. Bezerra

Abstract

MMOGs – massively multiplayer online games – have become, in the last decade, an important genre of online entertainment, having a significant market share. In these games, many thousands of participants play simultaneously with one another in the same match. However, this kind of game is traditionally supported by a powerful (and expensive) central infrastructure with considerable cpu power and very fast and low latency Internet connections. This work proposes the use of a geographically distributed server system, formed by voluntary nodes which help serving the game to the players. Two very important features, therefore, must be provided: consistency (which may be week and only existing in the players' perspective) and fault tolerance, as some of the participating nodes may crash and become unavailable. For this work, we consider the crash-stop failure model.

Research Advisor
Prof. Fernando Pedone

Research Co-advisor
Prof. Cláudio Geyer

Academic Advisor
Prof. Fernando Pedone

Review Committee
Prof. Committee Member1, Prof. Committee Member2

Research Advisor's approval (Prof. Fernando Pedone): Date: .....

PhD Director's approval (Prof. Michele Lanza): Date: .....

1 Introduction

MMOGs emerged in the last decade as a new trend because of the decrease of the domestic Internet connection cost and the increase of its average bandwidth. This kind of games have become very popular because they allow the online interaction of a massive number of players at the same time. In the most succesful cases, such as World of Warcraft [4], Lineage II [17] and EVE Online [5], for example, tens of thousands of players are supported simultaneously [6]. In these games, the players can interact with one another in a virtual environment.

Generally, each player controls an entity called *avatar*, which is his representation in the virtual environment – a character who executes the orders given by the player, therefore interfering in the outcome of the game. To provide a consistent view among the different participants, every action performed by each avatar is processed by a server, which calculates the game state resulting from such action. This new state is then broadcast to the players to whom it might be relevant – again, the state itself does not need to be consistent, but only what the players see.

To cope with the receiving and processing of thousands of actions and the broadcast of their respective resulting states, the traditional approach is to use a large central server – usually a cluster with a high-speed, low latency Internet connection – which provides such cpu power and network capabilities [9]. Decentralized alternatives could be used, so that such expensive infrastructure would not be necessary. For example, a fully decentralized system could be used, where each player is a peer responsible for calculating a portion of the game state and for synchronizing it with other peers [19, 10, 8, 11, 12].

In such peer-to-peer approach, the players would need to agree upon the outcome of the actions, in order for their view to be consistent. To reduce the traffic between peers, the state updates could be sent only to whom they are relevant. To define what is relevant for each player, the virtual environment could be divided into regions [20], so that a small peer group would be formed by players who were inside each of these regions. Players whose avatars were in a given region could form a smaller peer-to-peer group – possibly reducing also the cost of each agreement – and their own computers would decide the outcome of the players' actions in that area of the virtual environment. Unfortunately, however, the fully decentralized approach would imply a heavy communication cost on the players, specially when running some agreement protocol. This might not only degrade the quality of the game, but also increase the requirements imposed on the players' computational resources, therefore likely reducing their number.

We propose here, then, a system composed of geographically distributed nodes which may act as game servers – these nodes could be provided, for example, by volunteers or by companies which might have some commercial interest with the game being hosted. Some works, such as [2, 18, 7, 15] also propose the use of a distributed server, but they consider that the nodes are connected through a high speed and low latency network (e.g. a cluster), what makes their solutions only partially applicable in a scenario with highly dynamic and volatile resources, such as wide-area networks formed with volunteer resources. Such networks have some inherent problems: nodes with low availability and dependability, low bandwidth and low processing power compared to current dedicated MMOG servers maintained by large game companies.

The rest of this document is organized as follows: Section 2 gives an overview of the system model and describes briefly some of its protocols; Section 3 presents the research directions to be followed next and Section 4 lists some of the accomplishments achieved by this research so far.

2 Proposed model

We consider here persistent state real-time games with virtual environments where each player controls an avatar by issuing commands to it. Through the avatar, the player interacts with the virtual world and with objects present in it, such as avatars from other players. The basic operation of the game network support system must be as follows: when a player connects to the server, it must send the current state of the player's avatar and of the surroundings of its current location; after that, this player keeps receiving periodic state updates for the objects which are relevant to it (usually based on whether those objects may be seen by the player [3, 1, 16]). When a player issues a command (e.g. move his avatar from one place to another, pick up an object, attack another player's avatar etc.), it is sent as a message to the server, which processes it and decides its outcome, probably changing the state of the game, which is then broadcast to all involved players.

The next sections will describe the services the service must provide to the players connected to it (Section 2.1), the specific requirements for a fault-tolerant distributed MMOG server system, considering the crash-stop failure model, as well as the proposed distributed architecture (Section 2.2) and an algorithm devised to synchronize the game state between the replicas (Section **??**).

2.1 Services to be provided

As already mentioned, in order to provide the services required by the MMOG, this work proposes the use of a geographically distributed system composed of voluntary nodes working together to form a distributed MMOG server. This distributed server must deliver, at least, the following services:

- **Simulation** of the game, that is, the receiving of actions sent by the players and then processing their corresponding outcomes, which may result in changes to the game state;
- **Broadcasting** a portion of the current game state to the different players connected to it;
- **Storage** of the game state, that is, the combination of the states of each object in the virtual environment.

It is important to note that in real-time games (as opposed to turn-based games), due to the possibly fast-paced interaction of the players, it is critical to provide **timeliness** for the delivery of state update messages to them, so that they have the illusion of a common environment. Besides, it is necessary to provide **consistency** for the game state between each pair of players interacting with one another, which means that the game state they perceive must be as similar as possible. Although it is very hard to guarantee that every player will share the exact same view, it is important for their commands to be processed as if there was a single, strongly consistent game state, which is the one stored at the server.

2.2 Distributed server system

The main purpose of the system is to reduce or, ideally, eliminate the necessity of a central server for the game. In this work, the approach proposed to achieve this is based on principles similar to those of volunteer computing, where a potentially large set of contributors make their computers available to perform tasks requested by some remote entity. With such a system, although each of these nodes is probably much less powerful than a traditional mmog server, when working together they can sum up the cpu power and network link necessary to host one of these games.

However, several questions arise when dealing with a potentially large scale geographically distributed server system composed of voluntary nodes. First, as the nodes are volunteer, there is no guarantee that they will be available whenever they are needed, neither there is any guarantee regarding the amount of resources that each of these nodes makes available for the system – be it cpu power, network bandwidth or storage capability. Group communication may also be a problem, since primitives for that – such as network level multicast – are not widely available on the Internet.

Finally, considering the potentially large number of nodes in this system, it is very likely that some of them present failures, for what we are assuming the crash-stop failure model – therefore, we assume the possibility of nodes crashing and network links becoming unavailable. The services provided by the system must continue even in the presence of such failures. Not only must the distribution itself be transparent to the players, but also the existence of failures and their countermeasures must not be noticeable by them.

In Section 2.2.1, we describe how the game state will be distributed among the different server nodes; in Section 2.2.2, a preliminary consistency management protocol, based on state machine replication, is presented.

2.2.1 State partitioning

In order to split a game server into several smaller units executed by many volunteer nodes, it is used the idea of dividing the virtual environment in regions, each of which being managed by a different server. There are many approaches proposed in the literature, such as using fixed size cells which are grouped into regions, or using binary space partition trees, and so on. For the sake of generality, the architecture proposed here does not decide how exactly the partitioning of the virtual environment will be performed. To cope with the possibility of failures, each server has several replicas, forming a server group.

The state partitioning is performed based on what objects are located in each region. The state of a object is managed, then, by the server group assigned to the region where that object is located. This way, the locality of data is explored: objects closed to each other are more likely to interact and, as they are located on the same server group, their interaction should be faster. Besides, by exploring the locality of the game data (state of the objects), the inter-group communication overhead is reduced.

By having multiple server replicas in each region, the cost of broadcasting state update messages to the players is also reduced. Since all the replicas in a group are supposed to have an identical state, each one of them may send state updates to a different subset of the players located in that region. As the number of players may be as large as many thousands, this would probably be the most costly operation for the system and distributing it would have a significant impact on the system scalability.

2.2.2 Consistency protocol

One of the assumptions here is that the commands from the players lead to a deterministic resulting game state. In other words, if the game starts in the state S, there is only one possible state S' to be reached after a sequence $C = \{c_1, c_2, ..., c_n\}$ of commands. So, the game state may be modeled as a state machine, which is replicated across the different server nodes, as proposed in [13, 21, 14]. Therefore, a way to implement the distributed server system would be by creating such state machine and forward every command from the players to every server node in the same (total) order, so that they would always be able to calculate the correct game state[1], resulting in a fully replicated server system.

The problem with a fully replicated state machine approach for a distributed game server is that it would hardly scale well. As the number of servers increased, so would the cost of processing a command, as it would have to be totally ordered among other commands received by any of the other servers. Not only the number of control messages would grow and potentially saturate the servers' network bandwidth, but also the time needed for such total order to be achieved would possibly increase beyond a tolerable value.

We decided, therefore, to use a partial state machine replication scheme, so that only the server group responsible for a given state will have to process the commands that alter such states. To make this possible, each command must specify a target set – defined either by the player or by a server –, which consists of the objects whose states are needed or altered by the command. As each object is assigned to a different server group, the protocol will then be able to infer which servers must process such command.

Each command will be processed by the group which contains its targets, being ordered among other commands who also are relevant to that same group. When a command has targets from different groups, than it must be ordered in all of the groups involved, i.e. all the groups which manage one of that command's targets.

Optimistic multicast

3 Future work

4 Activities and publications

References

[1] D. T. Ahmed and S. Shirmohammadi. A Dynamic Area of Interest Management and Collaboration Model for P2P MMOGs. *Proceedings of the 12th IEEE International Symposium on Distributed Simulation and Real Time Applications, 2008, Vancouver, BC, Canada*, pages 27–34, 2008.

[2] M. Assiotis and V. Tzanov. A distributed architecture for MMORPG. *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, 2006.

[3] C. E. B. Bezerra, F. R. Cecin, and C. F. R. Geyer. A3: a novel interest management algorithm for distributed simulations of MMOGs. *Proceedings of the 12th IEEE International Symposium on Distributed Simulation and Real Time Applications, 2008, Vancouver, BC, Canada*, pages 35–42, 2008.

[4] Blizzard. World of Warcraft, 2004. http://www.worldofwarcraft.com/.

[5] CCP. EVE online, 2003. http://www.eve-online.com/.

[6] A. Chen and R. Muntz. Peer clustering: a hybrid approach to distributed virtual environments. *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, 2006.

[7] R. Chertov and S. Fahmy. Optimistic Load Balancing in a Distributed Virtual Environment. *Proceedings of the 16th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2006.

[8] A. El Rhalibi and M. Merabti. Agents-based modeling for a peer-to-peer MMOG architecture. *Computers in Entertainment (CIE)*, 3(2):3–3, 2005.

[1] It might be the case that some servers could still be processing an older command, while others might have already processed newer inputs. Although this might lead to an old state, it would never lead to an incorrect state. To check which one is the newest, a version number considering, for instance, the number of commands processed – since the commands arrive in the same order at the different servers – would be enough.

[9] W. Feng. What's Next for Networked Games? *Sixth annual Workshop on Network and Systems Support for Games (NetGames)*, 2007.

[10] T. Hampel, T. Bopp, and R. Hinn. A peer-to-peer architecture for massive multiplayer online games. *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, 2006.

[11] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. *Proceedings of ACM SIGCOMM 2004 workshops on NetGames' 04: Network and system support for games*, pages 116–120, 2004.

[12] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. *IEEE Infocom*, 2004.

[13] L. Lamport. The implementation of reliable distributed multiprocess systems* 1. *Computer Networks (1976)*, 2(2):95–114, 1978.

[14] B. Lampson. How to build a highly available system using consensus. *Distributed Algorithms*, pages 1–17, 1996.

[15] K. Lee and D. Lee. A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution. *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 160–168, 2003.

[16] R. Minson and G. Theodoropoulos. An adaptive interest management scheme for distributed virtual environments. *Proc. of PADS '05*, pages 273–281, 2005.

[17] NCsoft. Lineage II, 2003. http://www.lineage2.com/.

[18] B. Ng, A. Si, R. Lau, and F. Li. A multi-server architecture for distributed virtual walkthrough. *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 163–170, 2002.

[19] S. Rieche, M. Fouquet, H. Niedermayer, L. Petrak, K. Wehrle, and G. Carle. Peer-to-Peer-based Infrastructure Support for Massively Multiplayer Online Games. *Consumer Communications and Networking Conference, 2007. CCNC 2007. 2007 4th IEEE*, pages 763–767, 2007.

[20] G. Schiele, R. Suselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis. Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming. *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 773–782, 2007.

[21] F. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.