Università
della
Svizzera
italiana

Faculty
of Informatics

Research Prospectus

July 18, 2011

A fault-tolerant distributed infrastructure for massively multiplayer online games

Carlos Eduardo B. Bezerra

Abstract

Massively multiplayer online games (MMOGs) have become an important genre of online entertainment in the last decade. In these games, many thousands of participants play simultaneously with one another in the same match. This kind of application has been traditionally supported by powerful and expensive central infrastructures with considerable cpu power and efficient Internet connections. This work proposes a geographically distributed infrastructure to support MMOGs, formed by nodes that help serve the game to the players. The most important features that must be provided are: consistency, since different players should have the sense of one single shared virtual environment, timeliness, as these games are usually real-time applications, and fault tolerance, as some of the participating nodes may crash and become unavailable.

Research Advisor Prof. Fernando Pedone	Research Co-advisor Prof. Cláudio Geyer
Academic Advisor Prof. Fernando Pedone	Review Committee Prof. Kai Hormann, Prof. Nate Nystrom

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 successful cases, such as World of Warcraft [1], Lineage II [2] and EVE Online [3], for example, tens of thousands of players are supported simultaneously [4]. 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 the action. This new state is then broadcast to the players to whom it might be relevant.

To cope with the receiving and processing of thousands of actions and the broadcasting 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 [5]. Decentralized alternatives could be used, so that such expensive infrastructure would not be necessary. One example would be a fully decentralized system, where each player is a peer responsible for calculating a portion of the game state and for synchronizing it with other peers [6, 7, 8, 9, 10].

In such fully decentralized 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, each state update could be sent only to players to whom it is relevant. To define what is relevant to each player, the virtual environment could be divided into regions [11], so that players whose avatars were in a given region could form a small peer-to-peer group. This way, instead of all peers having to agree upon the outcome of every action, only the peers in a given group would have to agree upon actions which were relevant to that group. Unfortunately, this approach would still 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 reducing the number of potential participants.

We propose here 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 in hosting the game. Some works, such as [12, 13, 14, 15], also propose the use of distributed servers, 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 scenarios with highly dynamic and volatile resources, such as wide-area networks formed by volunteer nodes. 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 describes the basic features of an MMOG server; Section 3 introduces our proposed distributed infrastructure and describes briefly some of its protocols; Section 4 presents the research directions to be followed next and Section 5 lists some of the works we have published so far.

2 MMOG server overview

We consider here persistent state real-time games with virtual environments where each player controls an avatar – a virtual object that represents him – by issuing commands to it. Through the avatar, the player interacts with the virtual world and with other objects present in it, such as avatars from other players. The basic operation of the game infrastructure must be as follows: when a player connects to the server, the server 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 him – usually, objects are considered relevant to a player when he can see them [16, 17, 18]. When a player issues a command to the avatar (e.g., move from one place to another, pick up an object, attack another player's avatar), 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.

In order to give to the players the sense of a persistent virtual world through which they can wander and interact with each other, the MMOG server must provide, at least, the following services:

- **Simulation** of the game world, that is, the processing of commands sent by players to their avatars, calculating the corresponding outcomes, which may result in changes to the game state;
- **Communication** of the current game state to the different players, so that their view is consistent with the changes that might have happened;
- **Storage** of the game state, which is the combination of the states of all objects in the virtual environment.

It is important to note that in real-time games, as opposed to turn-based games, due to the possibility of fast-paced interactions, it is critical to provide **timeliness** for the delivery of state-update messages to the players, so that they have the illusion of a shared environment. For the same reason, 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 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.

3 Distributed infrastructure

The main purpose of the infrastructure is to reduce or, ideally, eliminate the necessity of a central server for MMOGs. The approach we propose 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 the nodes is probably much less powerful than a traditional MMOG server, when working together they can sum up the cpu power and the network bandwidth necessary to host one of these games.

However, several questions arise when dealing with a potentially large, 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 is there 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.

Finally, considering the potentially large number of nodes in this system, it is likely that some of them present failures. For simplicity, we assume the crash-stop failure model, where servers do not present Byzantine behavior. The services provided by the system must continue even in the presence of failures. Not only must the distribution itself be transparent to the players, but also the existence of failures and their countermeasures should not be noticeable by them.

In Section 3.1, we describe how the game state will be distributed among the different server nodes and, in Section 3.2, we present a preliminary consistency management protocol, based on state machine replication. We also introduce, in Section 3.3, our group communication protocol on top of which our consistency management will be developed.

3.1 State partitioning

In order to split a game server into several smaller units, executed by many volunteer nodes, we divide the virtual environment in regions, each of which managed by a different server. Different approaches can be found in the literature, such as dividing the space using fixed-size cells and grouping them to form regions [19], or using binary space partitioning trees [20]. To tolerate failures, each server has several replicas, forming a server group.

The state partitioning is performed based on which objects are located in each region. The state of an object is managed, then, by the server group assigned to the region where that object is located. This approach exploits data locality: objects close 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 exploiting the locality of the game data (state of the objects), the inter-group communication overhead is reduced.

As the number of players may be as large as many thousands, broadcasting state-update messages to them will probably be the most costly operation for the server system. By having multiple server replicas in each region, this cost will also be distributed: 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 whose avatars are located in that group's region.

3.2 Consistency protocol

As mentioned before, players interact by issuing commands and having them processed by servers. Servers calculate the resulting game state and send it to the players. Every command can alter the state of the game, possibly based on a previous state. We assume that commands lead to a deterministic sequence of game states. 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 is executed.

The game state could be modeled as a state machine, which would be replicated across the different server nodes, as proposed in [21, 22, 23], and each command would be responsible for a state transition. One way of implementing the distributed server system would be by creating such a state machine and forwarding all commands

to every server node, in the same (total) order. As the state transitions are deterministic, every server would be able to calculate the correct game state independently. We would, then, have 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 the total order of the commands 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 state. To make this possible, each command must specify a *target set*, defined either by the player or by a server, which contains the objects whose states are needed or altered by the command. As every object is assigned to a different server group, the protocol will then be able to infer which servers must process each command.

Instead of forwarding each command to every server, now only the servers that manage some object in the target set of a command would process it. The problem with this scheme is that, in order to execute some command, a group might need the previous state of an object managed by some other group. To solve this, a possibility would be to have the groups requesting the states they need from other groups. Another solution, probably more latency-efficient, would be every group proactively sending the new states of the objects managed by it to other groups which might need them. We intend to look into both alternatives.

3.3 Quasi-genuine multicast

To propagate commands to servers we are considering a multicast communication protocol. As commands seen by different servers must be handled in the same order, we need total order, so that the state machine replication approach can be used. Besides, the commands sent by each player should be executed in the same order in which they have been issued; therefore, the multicast protocol should provide FIFO ordering.

However, the inter-group communication overhead should be minimized, so that the distributed server would be more scalable. Non-genuine multicast protocols [24] require control messages to be exchanged between groups, even when there is no application level message to be sent. Genuine multicast algorithms minimize the number of messages exchanged, but they impose a greater delivery latency [25].

We devised a multicast protocol that, although not genuine, allows the system to increase its scale without getting flooded with control messages. We call it *quasi-genuine* because every group keeps sending messages to (and receiving from) other groups, but only a few of them. This is because the protocol takes into account communication patterns of the application. Such patterns can be inferred from the application semantics and given to the multicast algorithm as a parameter.

This communication primitive can be used by our consistency management algorithm because every server group only has to forward commands and states to a specific set of other groups, which are the ones managing neighbor regions; they are known as soon as the game partitioning has been performed. Changes to the partitioning should be made rarely, as it would incur in some players migrating between servers. Besides, changing the regions does not necessary change which groups can communicate with each other. Therefore, our quasi-genuine multicast protocol is well suited for the consistency management protocol we are devising.

To further decrease the message delivery time, an optimistic delivery is also being considered. If every server monitors the deviation between its clock and those of other servers, while also estimating the maximum message delivery time, it is possible to deliver messages in only one communication step and in total order. As these estimations may be incorrect, a second, conservative message delivery still must be done to confirm the optimistic one.

4 Future work

We are currently working towards the validation of the quasi-genuine multicast and the game state consistency management protocols, which require both correctness proofs and experimental results, to demonstrate their performance. Another direction that we intend to follow is to consider not only crash-stop, but also Byzantine faults. By tolerating these faults, the server system would be resistant to many kinds of cheating [26], which can also be seen as Byzantine behavior.

5 Published work

Some related works that we have published so far include:

- C. E. B. Bezerra, J. L. D. Comba, C. F. R. Geyer. A fine granularity load balancing technique for MMOG servers using a kd-tree to partition the space. *VIII Brazilian Symposium on Computer Games and Digital Entertainment - Computing Track*, 2009.
- C. E. B. Bezerra, C. F. R. Geyer. A load balancing scheme for massively multiplayer online games. *Multimedia Tools and Applications - Special Issue on Massively Multiplayer Online Gaming Systems and Applications*, v. 1, n. 1, pp. 263-289, 2009.
- C. E. B. Bezerra, F. R. Cecin, C. F. R. Geyer. A3: a novel interest management algorithm for distributed simulations of MMOGs. *12-th IEEE International Symposium on Distributed Simulation and Real Time Applications*, 2008.

References

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

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

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

[4] 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.

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

[6] 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*, pp. 763–767, 2007.

[7] 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.

[8] A. El Rhalibi and M. Merabti, "Agents-based modeling for a peer-to-peer MMOG architecture," *Computers in Entertainment (CIE)*, vol. 3, no. 2, pp. 3–3, 2005.

[9] 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*, pp. 116–120, 2004.

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

[11] 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*, pp. 773–782, 2007.

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

[13] 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*, pp. 163–170, 2002.

[14] 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.

[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*, pp. 160–168, 2003.

[16] 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*, pp. 35–42, 2008.

[17] 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*, pp. 27–34, 2008.

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

[19] B. De Vleeschauwer, B. Van Den Bossche, T. Verdickt, F. De Turck, B. Dhoedt, and P. Demeester, "Dynamic microcell assignment for massively multiplayer online gaming," *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pp. 1–7, 2005.

[20] C. E. B. Bezerra, J. L. D. Comba, and C. F. R. Geyer, "A fine granularity load balancing technique for MMOG servers using a kd-tree to partition the space," *VIII Brazilian Symposium on Computer Games and Digital Entertainment - Computing Track, 2009, Rio de Janeiro, RJ, Brazil*, 2009.

[21] L. Lamport, "The implementation of reliable distributed multiprocess systems* 1," *Computer Networks (1976)*, vol. 2, no. 2, pp. 95–114, 1978.

[22] F. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990.

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

[24] N. Schiper, P. Sutra, and F. Pedone, "Genuine versus non-genuine atomic multicast protocols for wide area networks: An empirical study," in *Reliable Distributed Systems, 2009. SRDS'09. 28th IEEE International Symposium on*, pp. 166–175, IEEE, 2009.

[25] N. Schiper and F. Pedone, "On the inherent cost of atomic broadcast and multicast in wide area networks," *Distributed Computing and Networking*, pp. 147–157, 2008.

[26] J. Yan and B. Randell, "A systematic classification of cheating in online games," *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pp. 1–9, 2005.