A Peer-to-Peer Architecture for Massive Multiplayer Online Games

Thorsten Hampel
University of Paderborn
33102 Paderborn
Germany
hampel@upb.de

Thomas Bopp
University of Paderborn
33102 Paderborn
Germany
astra@upb.de

Robert Hinn
University of Paderborn
33102 Paderborn
Germany
exodus@upb.de

ABSTRACT

Massive Multiplayer Online Games with their virtual gaming worlds grow in user numbers as well as in the size of the virtual worlds. With this growth comes a significant increase of the requirements for server hardware. Today an MMOG provider usually faces the problem of serving thousands of users with entire server clusters. Peer-to-Peer networks with their high scalability and flexibility meet the requirements of connecting hundreds of thousands of people all over the world without a central server. In doing so the network bandwidth requirements remain at a reasonable level.

In this work we propose to combine MMOGs with a Peer-to-Peer network. We introduce a game architecture capable of exploiting the flexibility and scalability of P2P networks. A P2P architecture based on an overlay network using distributed hash tables with support for persistent object storage and event distribution has been developed to meet MMOG requirements.

Keywords

MMOG, peer-to-peer, game architecture

1. INTRODUCTION

Distributed hash tables (DHT) are a mapping of a hash function to the nodes of a network. They route messages on shortest paths from one node to another (usually in a logarithmic number of routing steps) and guarantee fail safety of the overlay network. Obviously, this new technology offers a comprehensive peer-to-peer infrastructure for all kinds of applications. Therefore, the usage of peer-to-peer overlays for MMOG architectures is a promising approach.

In recent years the design of distributed hash tables has been studied extensively. Nevertheless, there are only a few applications based on this new technology. It is generally agreed upon the impact of DHT on the design of new software, when scalability, distribution of content, or computing power is an issue. Therefore, we adapt the concept to the field of massive multiplayer online games (MMOG). In this area scalability is an important factor when a lot of users play simultaneously in huge virtual worlds.

Peer-to-peer overlay networks can provide this kind of scalability, moving the computational load and storage requirements from central server clusters to peers in the network. This way, each player's computer connected to the game also automatically provides additional resources to the game. This reduces requirements for powerful central server clusters and thus lowers the cost of setting up and maintaining massive multiplayer online games. The main drawback of peer-to-peer networks for games is the lack of a central authority that regulates access and prevents cheating. We introduce a concept of sets of controller peers that supervise each other. This kind of redundancy can prevent cheating [3] while at the same time improving the stability of the network, since there is no single point of failure.

We will first present an overlay network based upon distributed hash tables (DHT) in a way usable for MMOGs and then present an MMOG architecture that we have implemented using such a peer-to-peer network.

2. CONCEPT

The concept of a P2P-based MMOG architecture offers a basic infrastructure to support a large number of participants. In order to meet the requirements of a distributed MMOG architecture we have chosen existing technologies and systems. Each of these systems can be mapped on certain requirements with some demands needing a combination of technologies. Our design is based on Pastry [5] and the extensions Past [6, 2] and Scribe [1]. The resulting overlay network includes the following components:

DHT-based Overlay-Network – Pastry

The DHT-based overlay-network should enable a self-organizing, virtual infrastructure, which offers robustness against network failures. Additionally, the infrastructure possesses high scalability to maintain good performance with a large number of peers. Communication and Persistence are handled by the infrastructure. In comparison to typical DHT-applications like filesharing the MMOG needs events for game actions and player communication. Pastry serves as a good basis for these requirements.

Object Management – Past

Netgames'06, October 30–31, 2006, Singapore.
Copyright 2006 ACM 1-59593-589-4. $5.00.

Pastry is responsible for maintaining the network and routing messages in the overlay network. An extension of the basic functionality is needed to achieve persistence of objects. It must satisfy the demands of a high availability with replication of objects. For this purpose the Past system is applicable, which is also based on the Pastry routing technology.

Event-based Messaging – Scribe

Scribe is a multicast infrastructure for Pastry. Therefore, it is ideally suited to distribute events (e.g. game actions) to the nodes of the peer-to-peer network. Apparently, this event infrastructure needs to be scalable as well. The delivery of messages is ensured because of Pastry's routing algorithm.

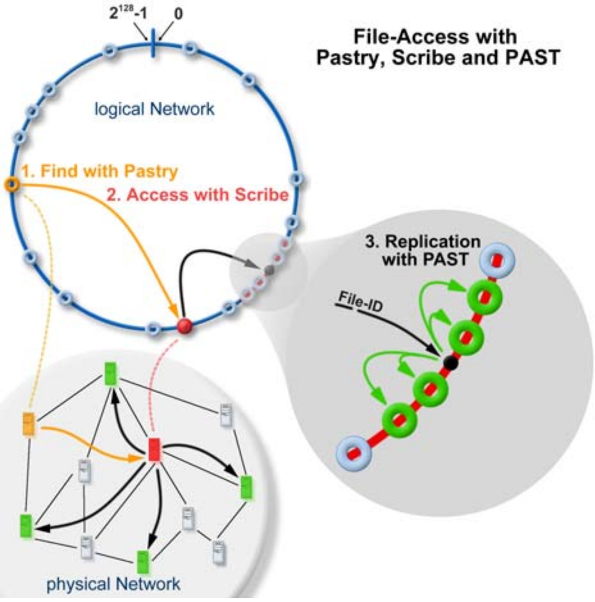

Figure 1: Combining different technologies to accomplish object access and synchronization

Figure 1 illustrates the cooperation of the Pastry-based technologies. In step 1 an object is found by Pastry itself. Then, all access is synchronized using the Scribe extension. Finally, the replica management is handled by Past by storing replicas to the currents node leaf set of the logical network. The illustration of the physical network shows how objects are replicated to provide redundancy and availability. The combination of these technologies covers the requirements of a distributed MMOG architecture. A detailed description of the implementation will be given in the next section.

3. MMOG ARCHITECTURE BASED ON A P2P OVERLAY NETWORK

To demonstrate the feasibility of a distributed overlay network approach, we have developed an MMOG architecture based upon the DHT overlay network described in the previous section. We use a Pastry network with persistent object storage through Past and event distribution through Scribe. The game engine that runs on each peer sitting on top of this network base is shown in figure 2.

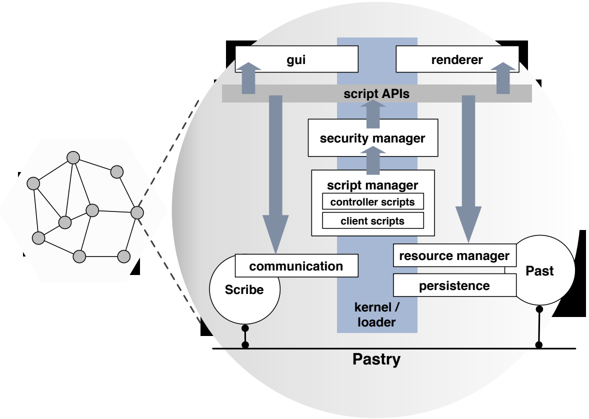

Figure 2: Architecture of a single peer node

As can be seen, this architecture resembles a microkernel architecture. The script and security managers form a kernel that controls inter-component communication through the scripting interfaces. The kernel also acts as a loader that fetches components from the overlay network through the resource manager. Central software updates can be realized by uploading new versions of components into the Pastry network and letting the resource manager load these at runtime into the environment of each peer.

To achieve a consistent timing for games realized with our architecture, we are separating the game runtime into distinct slots called 'ticks'. These can be arbitrarily defined, for a turn-based game it could be game turns, for real-time games it could be 20 millisecond phases. Each tick, a new game state is calculated from the previous game state and the changes that have arrived since then. The number of ticks since the beginning of the game can be used as a timing counter, so that each game action and game state can be attributed to a certain time stamp. This also makes it possible to determine synchronicity between peers.

We will describe the main components of our MMOG architecture in the following subsections.

3.1 Scripting and Security

The game-engine components are written in Java, while the game logic is implemented in the PNuts[1] scripting language. This allows for arbitrary games and for easy extendibility. In addition to that, all game components can provide a scripting interface so that there are clearly defined interfaces that result in code that is easier to maintain. A security manager handles access rights and permissions before passing script calls to the script manager for execution. In settings where different content providers write extensions for the game, this environment can be used to provide clear code interfaces and access control between components. It also limits

[1] http://pnuts.dev.java.net/

user actions to those that are allowed for the user within the game context.

3.2 Network

Several components provide functionality based directly on the overlay network (Pastry). The resource manager holds static data like script source code, images or 3d object data which are stored persistently in the overlay network (through Past). The data manager holds the game state and client specific data like login information and the user profiles and 3-d avatars. The communication manager handles event distribution by using multicasts on the overlay network (through Scribe). Events are routed through Scribe-topics and can be distributed and subscribed to just like in a centralized client-server architecture.

The game world is segmented into several regions. These regions can be chosen arbitrarily and they can be connected to provide a large environment as well as disconnected areas that are only accessible through doorways or portals. Each region is controlled by a region controller (RC), a peer that has been chosen to keep the game states of all other peers in the same region consistent. Figure 3 shows a part of the game world that is divided into hexagonal regions, each running on a Pastry network controlled by a region controller.

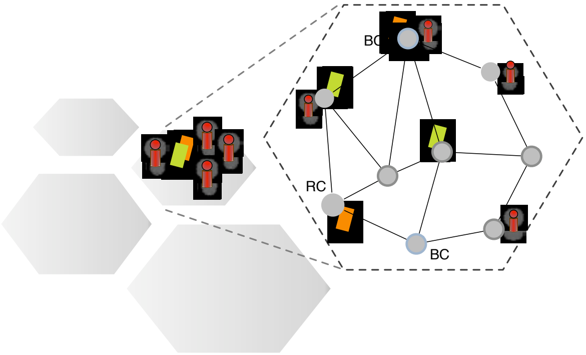

Figure 3: A region of the game world as an overlay network with redundant object storage and region controller

To increase network stability and tackle the problem of cheating by manipulated peers we introduce a number of backup controllers (BC). These keep copies of the data held by the region controller. In case the region controller fails or disconnects from the network, a backup controller can immediately take over and become the new region controller. The backup controllers can also detect when the region controller has been compromised and is trying to cheat by sending out manipulated game states (see figure 5).

3.3 Consistency and Events

One of the central problems of massive multiplayer online games is that of cheating, which means that a player tries to manipulate data that she is not entitled to change, e.g. the virtual wealth of the player's character in the game. In peer-to-peer architectures, this problem is even more sophisticated, because part of the virtual world and game logic is run on the player's peer node. This means that peers cannot be regarded as trustworthy and that special mechanisms must be developed to ensure that compromised peers cannot endanger the consistency or stability of objects and network traffic.

A central approach towards this problem is redundancy. Backup controllers can take over if a region controller fails and they can supervise each other and the region controller to detect and prevent cheating. The event and game state distribution in our peer-to-peer system works as follows:

1. Peers send events (e.g. game actions) to the region controller and the backup controllers.
2. The region controller and the backup controllers each calculate the new game state based upon the events they received. The region controller then sends out the new game state to all peers, while the backup controllers send out hash values calculated from the new game state.
3. The peers compare the new game state they received to the hashes from the backup controllers. If the hashes match, then the game state was valid. If they do not match, then they can determine which peer sent out an invalid game state or hash. This either means that the peer failed to calculate the correct hash or game state or that it has been compromised. A compromised node can thus be expelled from the network or reduced to common peer status depending on policy. If the game state was invalid, the peers skip until the next game state is sent. This might cause a small lapse in the game, but the game will not fail and will automatically be in sync again when the next valid game state arrives.
4. To prevent single peers from interfering persistently, the role of the region controller is passed around each game tick. This means that the load of calculating a game state and sending it out will be evenly distributed over time and no peer will have the chance to disrupt the game with invalid game states mutliple ticks in a row.

Figure 4 shows how the peers use the overlay network to distribute events or game actions between them. The distribution of game states by the region controller and game state hashes by the backup controllers can be seen in figure 5.

4. RELATED WORK

The presented approach is quite similar to the ideas introduced in [4]. SimMud is a proof-of-concept of an MMOG architecture based on Pastry and combining the Past and Scribe extensions to create a virtual world distributed to peers. It is shown that the system scales with the number of players with an average message delay of 150ms. Our approach provides a framework for MMOGs based on such a P2P architecture, while adressing some common game-related requirements like object and resource storage and basic mechanisms to prevent or hinder cheating.

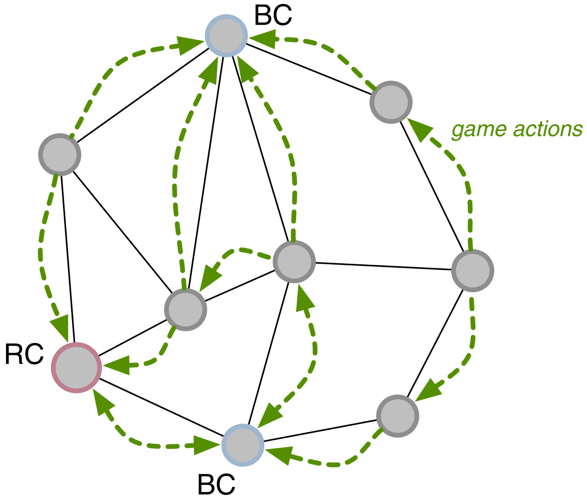

Figure 4: Game actions and events are sent to the region and backup controllers

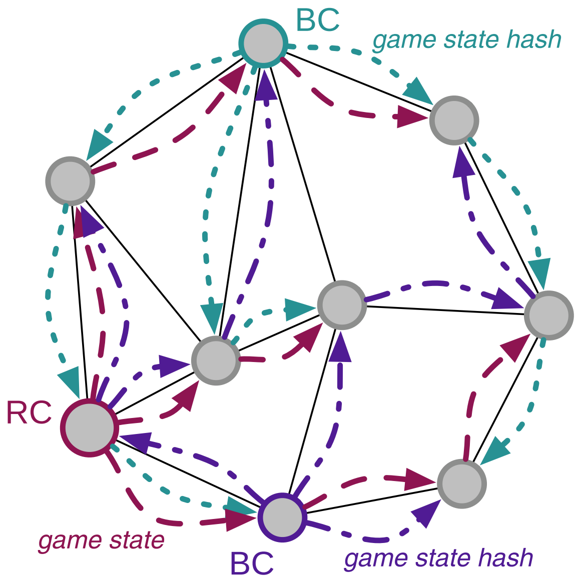

Figure 5: Distribution of game state and hashes for cheating detection

5. CONCLUSION AND OUTLOOK

Our architecture demonstrates the successful design of a highly distributed P2P-based gaming architecture by using Pastry in combination with the Past and Scribe extensions. A distributed MMOG is a very promising approach for the gaming industry, because the traffic and load can be moved to the former client machines.

The motivation for a distributed MMOG can be seen in a scalability of resources, robustness by decentralized storage of objects and management of peers, load-balancing by distribution of activities through the network, self-organizing network and efficient routing and replication.

Several details remain for further investigation. The latency of our example MMOG suffices for games that are not overly time-critical (e.g. a game world that is mainly used for communication and trade). We are planning to do more research on whether this architecture can also be used for games with higher requirements towards latency, e.g. action games.

6. ADDITIONAL AUTHORS

The MMOG student project group: Tobias Berghoff, André Braun, Kai Brinksmeier, Marco Gießman, Markus Heberling, Jürgen Hölker, Kamil Kopel, Marko Kowalczyk, Johannes Lintner, Ingo Niehaus, Markus Podlacha, Dilek Say, André Schmitz, Metaxia Vavritsa, Raphael Weber.

7. REFERENCES

[1] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489–1499, 2002.

[2] P. Druschel and A. I. T. Rowstron. PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility. pages 75–80, 2001.

[3] P. Kabus, W. W. Terpstra, M. Cilia, and A. P. Buchmann. Addressing Cheating in Distributed MMOGs. In *NetGames '05: Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 1–6, New York, NY, USA, 2005. ACM Press.

[4] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, 2004.

[5] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[6] A. T. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. pages 188–201, 2001.