FUTURE COLLABORATIVE SYSTEMS BETWEEN PEER-TO-PEER AND MASSIVE MULTIPLAYER ONLINE GAMES

Markus Heberling, Robert Hinn, Thomas Bopp
Department of Computer Science, University of Paderborn, Fuerstenallee 11, 33102 Paderborn, Germany

Thorsten Hampel
Department of Knowledge and Business Engineering, University of Vienna, Rathausstr. 19/9, A-1010 Vienna, Austria

Keywords: P2P, DHT, CSCW, software architecture.

Abstract: Most current CSCW architectures rely on a central server and offer only limited scalability. With the emergence of distributed hash tables as a comprehensive peer-to-peer infrastructure a wealth of new applications can be developed. In this paper we propose a new DHT-based CSCW architecture, inspired by modern massive multiplayer online games architectures, using existing systems and technologies. The resulting CSCW overlay network offers both robustness against network failures and scalability to support large numbers of users simultaneously.

1 INTRODUCTION

Today's collaborative systems are mainly based on client-server based architectures. People interact over various synchronous clients (chat, shared whiteboards, voice chat etc.) or asynchronous, typically web-based, clients with a central server. Naturally resulting out of the architecture model's limitations, its scalability is limited to a certain degree. This comes especially into the focus if larger number of users interact with a high density of user interaction. As another matter of fact single-server CSCW-architectures are also suffering the bottleneck of single server architecture regarding the question of security and stability. A single server coordinating, organizing and being the persistence layer for a number of clients forming a collaborative system is the critical component in any client-server-based CSCW-system.

Having these limitations in mind a close solution may be to apply multi-server or peer-to-peer approaches to collaborative systems. Multi-server CSCW-architectures are a difficult to implement architectural model, as collaborative systems are highly interactive systems, where users move from one server to another, share documents over the borders of a single server and build (link) structures of media-content from one server to another. All these technical behaviours should be hidden from the user, e.g. user transparency in a server-spanning way of interaction is an important issue. This high degree of user interaction makes replicated architectures, such as a distributed database approach, which is typical for large web-based systems, the wrong solution.

On the other hand modern peer-to-peer-architectures can be identified as an interesting alternative. They do not need a dedicated central server, nor are they limited to typical problems of scalability. Today's successful P2P-architectures (e.g. having Napster or the Groove toolkit in mind) at first can be described as hybrid client-server and P2P-architectures (even Napster needs central servers for sharing index-files for coordinating the peer-to-peer connections). Therefore, so called superpeers (Mizrak et al., 2003) serve different functions such as providing persistency of some parts of the distributed system or are able to guarantee for some security issues when setting up a trustable CSCW-system.

Existing peer-to-peer architectures, such as distributed hash tables (Rhea et al., 2005; Rowstron and Druschel, 2001b) can provide the basis for highly interactive collaborative systems.What we get is a highly scalable distributed persistence layer, where objects are distributed over the clients and objects may be distributed with a certain replication factor (unavailable peers, e.g. peers falling out of the network, do not endanger the integrity of the net-

Heberling M., Hinn R., Bopp T. and Hampel T. (2007).
FUTURE COLLABORATIVE SYSTEMS BETWEEN PEER-TO-PEER AND MASSIVE MULTIPLAYER ONLINE GAMES.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - ISAS*, pages 340-346
DOI: 10.5220/0002384403400346

work/collaborative systems.) What is missing are special features of such architectures resulting out of the collaborative scenarios of use (e.g. interactions of users/clients are spanning several peers in a collaborative system). Therefore we are mainly not only talking about a problem of setting up distributed persistency layers in a peer-to-peer architecture, but also dealing with server-spanning event management. Simultaneously, all other problems of collaborative architectures, such as user and group management have to be solved.

This paper presents a first step towards a next generation of peer-to-peer-based collaborative systems. Our main contribution is setting up a first infrastructure out of existing distributed hash tables components, which serves as a first good basis for peer-to-peer based collaborative systems.[1]

We propose our architecture and prototype as an open source solution based on Past and Pastry, which we shaped along the needs of collaborative systems. Secondly we present a first demonstrator, which shows the architecture's scalability through a massive multiplayer online game "snake" application. Our architecture therefore has close relations to massive multiplayer online games (MMOGs) (a relation which is in the tradition of our first classical CSCW-architectures based on multi user dungeons – MUDS). MMOGs share many of the requirements of collaborative systems and are in some degree even stronger in their demand for security, e.g. the so called "cheating" – attacks of some peers/players against the integrity of the network/game.

2 DHT PEER-TO-PEER NETWORK

Our approach for a peer-to-peer CSCW architecture is based upon existing technologies of distributed hash tables (DHT) for storage and communication, replacing traditional database and communication layers. Distributed hash tables store data on the distributed nodes of a peer-to-peer network, providing greater scalability and, given the use of redundant storage, better robustness through the lack of a single point of failure. We will present a peer-to-peer network based on the Pastry overlay network with the extensions Past for persistent storage within the network, and Scribe for message passing and event distribution on the network. In section 4 we will then use this network as a basis for a peer-to-peer architecture suited for CSCW systems.

Pastry (Rowstron and Druschel, 2001a) is a ring-based peer-to-peer overlay network. It provides efficient algorithms for localizing objects within the distributed hash table and for routing packets with the network. It is completely decentralized and self-organizing, inserting and removing nodes into the network automatically, thus allowing for easy scalability. Objects in Pastry are stored and referenced by a 128-bit id in a circular logical address range. Objects are stored on nodes with an id closest to the object's id, so that the hash function of Pastry can localize them within a small number of hops. Nodes provide their own routing tables to their logical neighbors and are thus able to route network packets to the correct destination. These routing tables are updated whenever neighboring nodes leave the network, so that the network reorganizes itself automatically. This means that the routing algorithms in Pastry are able to cope with node failures or a dynamic participation of nodes.

Past (Rowstron and Druschel, 2001b) is an extension of Pastry for persistent storage of objects within the overlay network as a DHT. While Pastry manages the nodes of the overlay network, Past offers functions to insert, lookup and retrieve objects on the peer nodes. To achieve a high availability and robustness against failures in a dynamic peer-to-peer network, objects are replicated and the replicas are stored on adjacent nodes. Since adjacent ids in the logical address range usually do not represent peers that are adjacent in the underlying physical network, this leads to a wide distribution of replicas over physically remote nodes. This reduces the risk of objects not being available when several neighboring nodes leave the network.

Scribe (Rowstron et al., 2001) is a decentralized multicast infrastructure based on Pastry. It provides the functionality of a publisher/subscriber model for messages and events. This extends Pastry with a very flexible and efficient event distribution system. The multicast-trees in Scribe reorganize themselves when nodes leave or join the network. With the same replication mechanisms as in Past, this leads to the same stability and flexibility.

3 REQUIREMENTS FOR A PEER-TO-PEER CSCW ARCHITECTURE

We will describe a set of requirements that a peer-to-peer based CSCW architecture must fulfill to provide a usable basis for distributed collaborative systems.

[1]The authors have long lasting expericences in the design of collaborative systems. (Bopp et al., 2006)

A distributed CSCW architecture is expected to offer the same stability and much of the same functionality that traditional client-server architectures provide. However, they should also provide much of the same basic functionality of established client-server CSCW architectures, therefore we will present a basic set of such requirements.

Since virtual knowledge spaces have proven to be a solid and flexible concept for collaborative systems architectures (Bopp et al., 2006), we will also propose them as a basis for a peer-to-peer based architecture. A sophisticated user and permissions management is a necessity for collaborative systems as is a mechanism for dispatching and receiving events or notifications within the system. We will describe these central requirements in greater detail in the following sections.

3.1 Virtual Knowledge Spaces

Virtual knowledge spaces ("rooms") are persistent containers for documents, users and other objects, but also act as nodes for synchronous as well as asynchronous communication and collaboration. They form the basis for continuous, location- and time-spanning cooperation. Since rooms can be interlinked with exits and can contain other rooms they can be used to create structures or hierarchies, thus providing a solid base for individual and collaborative structuring of knowledge.

The representation of rooms can open up their functionality in different ways, e.g. a chat program could access a room as a communication channel, a file browser could access the room like a virtual folder in the filesystem, while at the same time a web browser could list the users and documents contained in the room. All these custom views work on the same data, but can provide functionality and representation tailored to their use. If virtual knowledge spaces are implemented in a software that also provides standard protocols like HTTP, WebDAV, FTP, IRC or XMPP, then this would enable users to use their preferred client software also for their collaborative work.

The functionality of custom views can also be tailored to specific roles or tasks. A room could provide a view for students where they can submit assignments and are able to view and change their own assignment up to a certain deadline. A different view for tutors could then list all submitted assignments and provide means to comment on them and rate them, which in turn would be accessible only by the students that submitted that assignment.

3.2 User and Permissions Management

Functionality like this obviously requires a sophisticated permissions management. Users should be assignable to groups that can be used to organize them but also to simplify access rights by considering groups as roles (e.g. guest, student, tutor, administrator, etc.). Groups should also be able to contain sub-groups, so that flexible or hierarchical user management becomes possible. Access Control Lists stored on objects and rooms can provide a good basis for permissions management, although the common "read", "write" and "execute" permissions known from filesystems are not differentiated enough for CSCW systems. A finer granularity of permissions is needed, e.g. "change content", "delete", "move" and "insert" instead of just "write". One aspect that truly unfolds the possibilities of virtual knowledge spaces is the inheritance of permissions from other objects, e.g. the environment that an object is contained in. This makes taking a document and putting it into a different room possible without having to change the permissions afterwards to allow the users with access rights to the target room to also access the document that has been placed there. This allows a very intuitive and at the same time very sophisticated and flexible permissions management on the basis of virtual knowledge spaces. An additional permission flag that opens up new usage scenarios is the right to delegate permissions to other users. This "sanction" permission provides a mechanism for self-administration where users who have the "sanction" permission on an object can delegate the other permissions they have on the object to other users.

3.3 Event System

To support awareness mechanisms in synchronous clients, the architecture must also support the creation, subscription and distribution of events between different components and, in the case of a distributed architecture, between different peers. More generally speaking, events can form the basic means of interaction between components of the software, providing a flexible infrastructure that additional components can easily interface with.

4 PEER-TO-PEER ARCHITECTURE FOR CSCW

Based on the overlay network described in section 2 and realized as a distributed multiplayer game architecture (see section 5) and the CSCW concepts of sec-

tion 3, we will now describe a peer-to-peer architecture for CSCW systems.

4.1 Basic Network Layers

For an easier to maintain architecture, the proposed CSCW system doesn't implement the peer-to-peer network functionality itself. Instead the application is divided into abstraction layers, as shown in figure 1: the peer-to-peer overlay network is layered on top the physical network, a persistence layer provides access to the network resources and event routing in a more generic way and presents this functionality to the CSCW system itself. This keeps the overlay network code out of the main application and would even allow exchanging the DHT implementation later on.

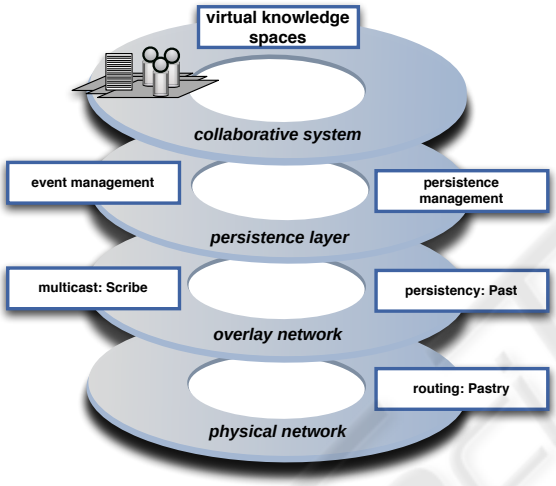

Figure 1: Basic layers of a distributed CSCW architecture.

Our architecture uses Pastry, Past and Scribe for the overlay network layer and implements a persistence layer for persistent object storage and event routing on top of it. Higher level functionality, like access rights management, are then based upon this object model and are located in the collaborative systems layer.

4.2 Distributed Virtual Knowledge Spaces

Virtual knowledge spaces inherently subdivide the persistence space into smaller segments, rooms. Figure 2 shows how virtual knowledge spaces are realized as sub-networks within the Pastry network. We map these rooms to Scribe multicast groups to achieve a smaller overlay network within the whole Pastry network. Communication within a knowledge space is therefore done by multicasts inside the group. This allows us to route events based on objects within a room only to those nodes that are registered to receive events for those objects, concentrating the main bandwidth demand of intense, synchronized user interaction to the network connections between those peers that take part in that interaction. This means that users collaborating within a shared whiteboard and using voice or video chat will not flood the network connections between other peers within the same collaborative system.

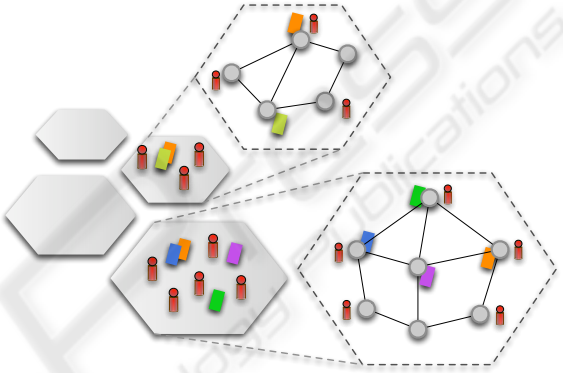

Figure 2: Virtual knowledge spaces in a Pastry overlay network.

Figure 3 shows how an action like "give a copy of a document to another user" is performed in a distributed collaborative system. A copy of the document is created and referenced by the target user's inventory. The document is not necessarily placed on the same peer node as the user or the original document, but it can be referenced through the routing of object ids in Pastry. Since the document inherits the access rights of its environment by default, the target user can then access it or place it in another room, e.g. her personal workspace or a group workspace.

To synchronize the communication within a group, we introduce controller peers that act as a server-like entity. The state of all objects inside the knowledge space is managed by this controller peer. If a peer wants to change the state of an object, it requests this change by sending a message to the controller peer, which processes this request. The controller peer then multicasts the changes to the other members of the group. For higher security and stability we introduce backup controllers, that monitor the actions of the controller and can take over in case of failure or misbehavior of the controller peer.

The region controller and the backup controllers need (and should) not be peers that are involved in

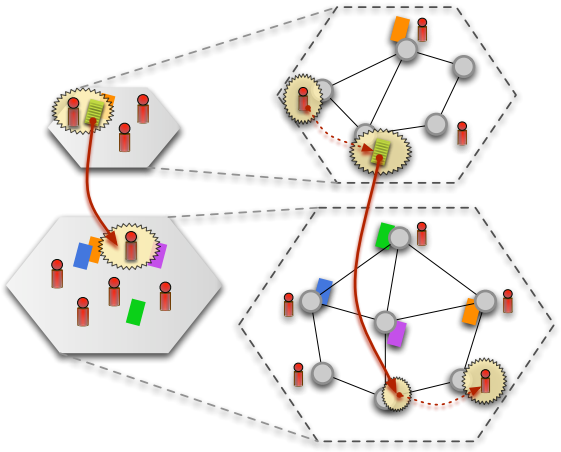

Figure 3: Actions across knowledge spaces: region-spanning event distribution.

that room themselves. This reduces the chance of manipulated peers to deliberately access or modify data. Since the region controller is supervised by the backup controllers, all these peers would need to be compromised in order to be able to "cheat" objects they control. These controller peers can be regarded as a first interface where enhanced security mechanisms can be integrated in the future.

4.3 Connecting to the Overlay Network

To join a peer-to-peer network some kind of bootstrapping is needed. This means the user needs an address to connect to, from which all necessary network information is delivered to the new node, so that it can be integrated into the network. Depending on the usage scenario different methods are available. In a local environment an architecture like Bonjour / Zeroconf can be used to discover nodes that offer a bootstrap service for the overlay network. In closed systems, where new users have to be invited to join the network, the bootstrapping info can be attached to the invitation message.

In a pure peer-to-peer setting, a user account is created by producing a public/private encryption key pair, storing the public key in the overlay network under the name or login name of the new user. To authenticate, a user would later send a message signed with her private key into the overlay network. This message can be verified with the public key to authenticate the user.

In all other scenarios some rendezvous server has to be established. This server would have a static network address, that is known by new nodes and could also offer authentication or accounting services if required, accessing central LDAP or Kerberos servers for example. Such central servers could also offer services that are traditionally provided under static addresses, like HTTP for the publishing of websites, FTP or WebDAV for file transfer, IRC or XMPP for chat, etc. It would also be possible to run such a central peer node inside a webservice container like Apache Tomcat, so that it can provide functionality of the distributed collaborative system to traditional webservice clients through the SOAP protocol.

The main drawback of using central "superpeer" nodes is that they will suffer high bandwidth requirements and be single points of failure. On the other hand, a number of superpeers can also be used as replication nodes for important data, thus guaranteeing the availability of certain data even in the unlikely case that all other peers with replicas of that data are disconnected from the network.

4.4 Searching

While looking up objects by their id is handled by Pastry, searching for resources in peer-to-peer networks is a non-trivial task that is mostly done by either introducing superpeer nodes that index resources or by plain broadcasting of search queries into the overlay network. Both approaches have shortcomings though. With superpeer nodes you have single points of failure. If such a node fails, large parts of the network become unsearchable. On the other hand broadcasting wastes bandwidth and processing power on all nodes (Charwathe et al., 2003).

We propose a different solution for this problem, by creating Scribe groups for each search word. If a node has a resource, that should be available by searching, it joins the Scribe group for all relevant search words. If another node wants to access all resources related to a given topic, it publishes a message to the Scribe group of that topic. The query is then multicasted to all members of that group. Since all group members have documents that are relevant to that topic, they send a message back to the original sender with the requested information. This simple algorithm obviously works only for single search words, but can be easily extended to support complex querys including AND and OR constructs.

To improve search results further we use tagging functions similar to popular web applications like Flickr[2], which allow users to tag resources. These tags can then be searched by the system.

[2]Flickr is an online photo community where users upload photos and tag them with keywords: http://flickr.com

5 DEMONSTRATOR – A PEER-TO-PEER MASSIVE MULTIPLAYER ONLINE GAME

To demonstrate our approach we implemented a massive multiplayer online game (MMOG) version of the classic snake game (figure 4). The game world is divided in regions and each region is assigned to a controller peer. The snake controlled by a player is always in one of these regions. The player can enter neighboring regions by crossing the screen border. If he/she does so, his/her snake is assigned to the new region and is deleted from the old region. Currently it is not possible for a snake to gradually change to a region piece by piece. To speed up the region change, the player node not only subscribes to the Scribe group of the current region it is in, but also to all neighboring regions. Upon region change, the new region is already joined and the new neighbors can be joined in the background, while the game is continuing. This behavior is not needed in CSCW systems, since there is usually no need for fast region changes. We currently send the game state of each region every 100 milliseconds, which is an update rate not needed for most CSCW applications. Synchronous clients like shared whiteboards might benefit from high update rates to provide smooth, real-time cooperation. The architecture is described in detail in (Hampel et al., 2006).

We've successfully run 128 instances (8 computers, each running 16 instances) of the snake game to measure stability and bandwidth usage (figure 5). The figure shows the average bandwidth used by a peer in the network in kbytes per second with a fixed number of 9 regions. We have found that the average bandwidth usage does not depend on the number of peers, but rather on the number of active regions. This is shown in figure 6, where we measured increasing region numbers with a fixed peer number of 8. The average bandwidth usage is quite high in our test settings because of our high update rate and the fact that each player not only joins the region his snake is in, but also all neighboring regions. This would not be the case in a CSCW-system since a peer only has to join the region the user is currently in.

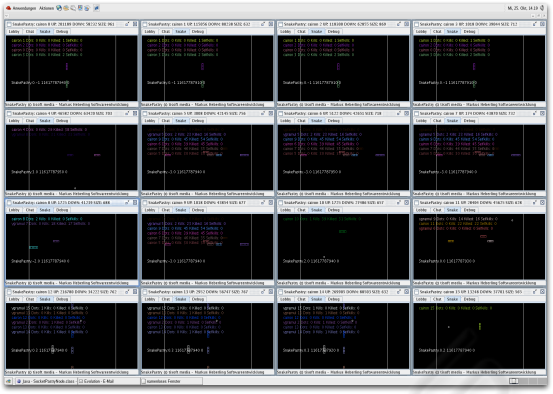

Figure 4: Screenshot of MMOG-Snake with 16 instances.

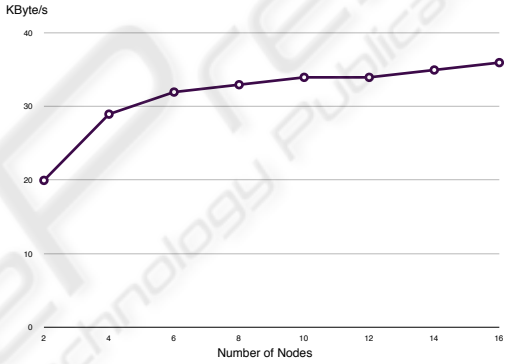

Figure 5: Average bandwidth usage with 9 regions and increasing peers.

6 RELATED WORK

SimMud (Knutsson et al., 2004) is a proof-of-concept massive multiplayer online game architecture. It is also based on Pastry as a distributed hash table overlay network and the Past and Scribe extensions. SimMud has shown that such a system scales well with large numbers of users even in synchronous settings.

OceanStore (Kubiatowicz et al., 2000) is a concept for a distributed filesystem, which focuses on the availability of data. This is accomplished by distributing the data on the nodes of an overlay network based on Pastry. All data is secured with cryptographic algorithms and stored redundantly to guarantee high availability.

OpenDHT (Rhea et al., 2005) is a service implementation of a distributed hash table. It provides a simple interface to retrieve and store data. This could qualify it as a persistence layer for CSCW systems.

Groove[3] is a hybrid architecture, where a central server provides an entry point into the distributed network of peers and also stores changes for peers who are offline. The main focus of Groove is support for decentralized, synchronous collaboration.

[3] www.groove.net

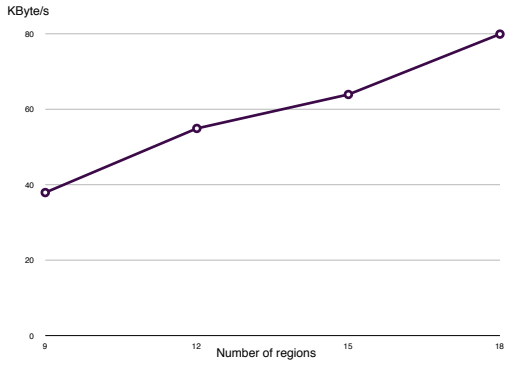

Figure 6: Average bandwidth usage with 8 peers and increasing regions.

7 CONCLUSION AND OUTLOOK

A peer-to-peer architecture for CSCW systems is feasible. Our demonstrator application shows that network bandwidth only increases logarithmically with the number of connected peers. This simulation was a synchronous, highly interactive setting which can be regarded as a "'worst case"' scenario for synchronous collaborative work, e.g. with a shared whiteboard. Most actual uses will have much less frequent event messages, resulting in even less bandwidth requirements. A peer-to-peer-based architecture is therefore highly scalable even for synchronous collaboration between large numbers of users. Since storage space and processing power are supplied by all peers, these resources even grow with increasing numbers of connected peers, while replication strategies ensure availability of data even in case many peers disconnect.

Our architecture is based upon virtual knowledge spaces on a peer-to-peer persistence layer. This permits distributed collaborative systems complying with the basic requirements for collaborative architectures, such as persistent object storage and an event distribution mechanism.

There are, however, aspects that still need to be investigated in greater detail. Security is one of the main concerns. A peer-to-peer CSCW framework must guarantee that data is only accessible by those that are entitled to it through the system's permission management. Since the functionality of the system is not confined within a single server, there must be reliable mechanisms that prevent a compromised peer to bypass access management of the software. A first approach is the use of region and backup controllers, but more sophisticated security measures should be researched.

REFERENCES

Bopp, T., Hinn, R., and Hampel, T. (2006). A service-oriented infrastructure for collaborative learning in virtual knowledge spaces. In Kumar, D. and Turner, J., editors, *Education for the 21st Century – Impact of ICT and Digital Resources*, volume 210, pages 35–44. International Federation for Information Processing, Boston: Springer.

Charwathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., and Shenker, S. (2003). Making gnutella-like p2p systems scalable. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 407–418. ACM Press.

Hampel, T., Bopp, T., and Hinn, R. (2006). A peer-to-peer architecture for massive multiplayer online games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games NetGames '06*.

Knutsson, B., Lu, H., Xu, W., and Hopkins, B. (2004). Peer-to-peer support for massively multiplayer games. In *Proceedings of the Conference on Computer Communications (INFOCOM)*.

Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. (2000). Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 190–201. ACM.

Mizrak, A., Cheng, Y., Kumar, V., and Savage, S. (2003). Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In *Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP)*, pages 104–111, San Jose, CA, USA.

Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., and Yu, H. (2005). Opendht: a public dht service and its uses. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 73–84, New York, NY, USA. ACM Press.

Rowstron, A. and Druschel, P. (2001a). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350.

Rowstron, A., Kermarrec, A.-M., Castro, M., and Druschel, P. (2001). Scribe: The design of a large-scale event notification infrastructure. In *Proceedings of the Conference on Networked Group Communication*, pages 30–43.

Rowstron, A. I. T. and Druschel, P. (2001b). Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Symposium on Operating Systems Principles*, pages 188–201.