A Storage and Indexing Framework for P2P Systems

Adina Crainiceanu Prakash Linga Ashwin Machanavajjhala
Johannes Gehrke Jayavel Shanmugasundaram

Cornell University, Department of Computer Science
{adina,linga,mvnak,johannes,jai}@cs.cornell.edu

ABSTRACT

We present a modularized storage and indexing framework that cleanly separates the functional components of a P2P system, enabling us to tailor the P2P infrastructure to the specific needs of various Internet applications.

Categories and Subject Descriptors

H.2.4 [**Database management**]: Systems—*distributed databases*

General Terms

Design

Keywords

peer-to-peer, p2p, indexing framework

1. INTRODUCTION

On the Internet, there are many applications that benefit from the cooperation between peers. These applications range from simple file-sharing to robust Internet-based storage management to digital library applications. Each of these applications imposes different requirements on the underlying peer-to-peer (P2P) infrastructure. For example, file-sharing applications need equality and keyword search capabilities, but do not need sophisticated fault-tolerance. On the other hand, storage management requires only simple querying, but requires robust fault-tolerance. Digital library applications require both complex queries, including equality, keyword search, and range queries, and sophisticated fault-tolerance. Other applications, such as service discovery on the Grid, impose their own specific requirements on the underlying P2P infrastructure.

One solution to this problem is to devise a special-purpose P2P infrastructure for each application. Clearly, this is quite wasteful, and does not leverage the common capabilities required across many applications. We propose a modularized P2P system architecture that cleanly separates different functional components, and allows us to reuse existing algorithms and tailor the system to the needs of the application. Here, we focus on the storage management and indexing aspects of the architecture, with the following components:

1. Fault Tolerant Torus: Provides fault-tolerant connectivity among peers.
2. Data Store: Stores actual data items and provides methods for reliably exchanging items between peers.
3. Replication Manager: Ensures that data items are stored reliably even in the face of peer failures.

Copyright is held by the author/owner(s).
WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

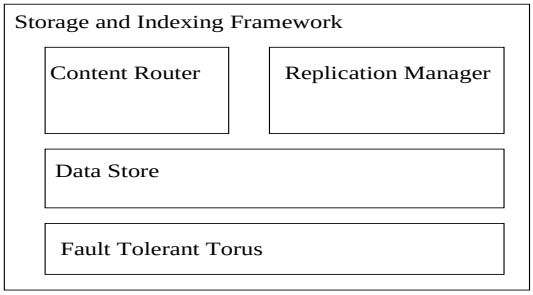

Figure 1: P2P Storage and Indexing Framework

4. Content Router: Allows efficient location of data items.

An additional benefit of the above framework is that we can use *existing* algorithms proposed in the literature for the various system components, and devise an overall system whose functionality is greater than any existing P2P system that we are aware of. Specifically, we can use the Chord [2, 7] algorithms for the Fault-Tolerant Torus and Replication Manager, the PePeR [3] algorithms for the Data Store, and the Skip Graph [1] algorithms for the Content Router, to create a system that is fault-tolerant, supports both equality and range queries, provides logarithmic search performance, and supports possibly large sets of items per peer. We are not aware of any other existing system that supports all of the above functionality in a P2P environment, even though each of the individual components by itself is not new.

2. THE FRAMEWORK

Figure 1 shows the components of our storage and indexing framework. We now provide an overview of these components and examples of their instantiation. We do not discuss each component's exact API due to space limitations.

2.1 Fault Tolerant Torus

The primary goal of the Fault-Tolerant Torus (FTT) is to maintain the connectivity of the peers in the system. Conceptually, the FTT maintains a mapping of regions in a torus to peers in the P2P system. We say that a peer is responsible for the region(s) assigned to it. The regions have to be contiguous, so all the space is covered, and non-overlapping, so any point on the torus is mapped to a single peer. The exact torus space and the method of maintaining the mapping depends on the particular implementation of the FTT.

Example Figure 2 shows an example of a ring (torus of dimension 1) and a mapping of ranges (regions) to peers. Peer p_1 is responsible for the range $(5, 10]$, p_2 is responsible for the range $(10, 15]$ and so on. Each region of the ring

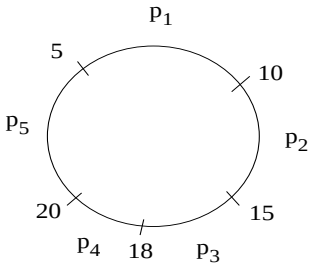

Figure 2: Mapping Ranges to Peers

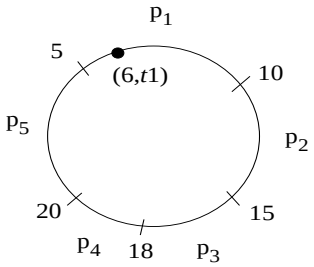

Figure 3: Mapping Values to Peers

space is mapped to one (and only one) peer. Now, assume that peer p_1 fails. In this case, the Fault Tolerant Torus needs to reassign the range $(5, 10]$ to another peer. If a peer can be responsible for only one region of the space, then p_2 or p_5 need to increase their range by taking over p_1's range.

The ring in Chord [7] is one possible instantiation of the Fault Tolerant Torus. Here, the integer space $[0, 2^m)$ is the ring space and peers are assigned an ID in this space. Each peer is responsible for the region in the ring between its predecessor ID and its ID. Other examples are the ring in Pastry [6] and the d-dimensional torus in CAN [5].

2.2 Data Store

The Data Store is responsible for distributing the data items to peers. Ideally, each peer should store approximately the same number of items, achieving storage balance. The Data Store maps each data item to a point in the torus space, and stores the item at the peer responsible for the region containing that point. If a peer ends up with too many data items (due to insertions) or too few data items (due to deletions), it will have to re-balance the assignment of data items to peers. Exactly how this re-balancing is done depends on the specific instantiation of this component.

Example In Figure 2, assume that a data item $t1$ mapped to value 6 is inserted into the system. In this case, the pair $(6, t1)$ will be stored at peer p_1 as shown in Figure 3.

The equivalent of the Data Store in Chord is implemented using a hash-based scheme. Data items are hashed to values on the ring, and assigned to the first peer with ID following the value in the ring (note that since hashing destroys the ordering of the values, Chord cannot process range queries). PePeR [3] does not use hashing, but maps data items to the peers responsible for the respective ranges in the value space. Re-balancing is done at peer insertion or deletion/failure, when some ranges are split respectively merged.

2.3 Replication Manager

The Fault Tolerant Torus component is responsible for ensuring that each point on the torus is assigned to some peer and the Data Store component is responsible for actually storing the data items at peers. However, if a peer fails, the data items it stored will be lost even if another peer takes over the "failed" region. The role of the Replication Manager is to ensure that all the data items inserted into the system are reliably (under reasonable failure assumptions) stored at some peer until the items are explicitly deleted.

Example In Figure 3, peer p_1 stores the pair $(6, t1)$. If p_1 fails, peer p_2 or p_5 will take over the range $(5, 10]$ (as ensured by the Fault Tolerant Torus component). However, the data item $t1$ would be lost. If the pair $(6, t1)$ is replicated at another peer in the system, the data item can be recovered.

The Replication Manager can be instantiated using the techniques proposed in CFS [2], where a peer's items are replicated to its successors in the ring, or PAST [4].

2.4 Content Router

The Content Router is responsible for efficiently routing messages to their destination in the P2P system. This component implements the search primitives, such as equality and/or range queries. The Content Router component could be instantiated using the Chord finger tables, the CAN neighborhood table, or Skip Graphs [1].

3. APPLICATIONS OF THE FRAMEWORK

One of the main applications of the framework is to tailor the system to the needs of the application. For example, an Internet-based P2P file sharing system could use the Fault-Tolerant Torus (for connectivity), the Data Store (for managing files), and a somewhat sophisticated Content Router (for supporting equality lookups and keyword search queries), but has no need for the Replication Manager. An Internet storage management system, on the other hand, needs a Replication Manager (for reliable storage), but only needs a simple Content Router (only equality name lookups), while the other components are the same as in the file sharing system. P2P digital library systems, however, need a very sophisticated Content Router (that supports equality, keyword search, and range queries), a robust Replication Manager and so on. The main benefit of the framework is that it allows us to plug in the appropriate instantiation of the relevant components for each application, without having to redesign the entire system from scratch.

As described in the introduction, the other benefit of our framework is that it enables us to put together a system that has more functionality than any existing system, solely by using *existing* components. Such a system is ideally suited for the digital library application described above.

4. CONCLUSION

We have proposed a modularized storage and indexing framework for P2P systems. This framework allows the reuse of existing algorithms for various applications with differing requirements. The framework also allows us to put together novel systems using existing components. We have a preliminary implementation of our framework, and are currently experimenting with different instantiations for digital library, resource discovery, and file-sharing applications.

5. REFERENCES

[1] J. Aspnes and G. Shah. Skip graphs. In *SODA*, 2003.

[2] F. Dabek et al. Wide-area cooperative storage with CFS. In *SOSP*, 2001.

[3] A. Daskos et al. Peper: A distributed range addressing space for p2p systems. In *DBISP2P*, 2003.

[4] P. Druschel et al. PAST: A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, 2001.

[5] S. Ratnasamy et al. A scalable content-addressable network. In *SIGCOMM*, 2001.

[6] A. Rowstron et al. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[7] I. Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.