An Indexing Framework for Peer-to-Peer Systems

Adina Crainiceanu, Prakash Linga, Ashwin Machanavajjhala
Johannes Gehrke, Jayavel Shanmugasundaram
Cornell University
Computer Science Department
adina,linga,mvnak,johannes,jai@cs.cornell.edu

1. INTRODUCTION

Current peer-to-peer (P2P) indices are monolithic pieces of software that address only a subset of the desired functionality for P2P databases. For instance, Chord [6] provides reliability and scalability, but only supports equality queries. Skip Graphs [1] support equality and range queries, but only for one data item per peer. PePeR [4] supports equality and range queries over multiple data items per peer, but does not provide any search or reliability guarantees in face of multiple failures. Galanis et al. [5] describe an index structure for locating XML documents, but this index does not provide any provable guarantees on size and performance.

In a P2P database system, all of the above functionality is required, but none of the existing systems supports it. We devise a modularized indexing framework that cleanly separates different functional components. This allows us to reuse existing algorithms rather than implement everything anew and to experiment with different implementations for the same component so that we can clearly evaluate and quantify the benefits of a particular implementation. Our indexing framework has the following components:

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

Based on this modular framework, we will demonstrate P-Ring, a novel index structure that supports equality and range queries, is fault-tolerant, gives guaranteed logarithmic search performance in a consistent system, and supports possibly large sets of items per peer. We are not aware of any other existing index structure that supports all of the above functionality in a dynamic P2P environment. We will also demonstrate two existing P2P index structures proposed in the literature, Skip Graphs and Chord [1, 6], implemented in the context of our framework. Our demo will illustrate the tradeoffs between the different index structures.

SIGMOD 2004 June 13-18, 2004, Paris, France.
Copyright 2004 ACM 1-58113-859-8/04/06 . . . $5.00.

2. SYSTEM ARCHITECTURE

We now provide a short overview of our indexing framework components shown in Figure 1.

2.1 Fault Tolerant Torus

The primary goal of the Fault Tolerant Torus (FTT) is to provide reliable connectivity among peers. This is important in a P2P setting where peer and network failures can occur at any time. Conceptually, the FTT implements a mapping of convex regions in a torus of search key values to peers in the P2P system. We say that a peer is *responsible* for the region(s) assigned to it. Regions are selected such that they are contiguous and non-overlapping, so any point on the torus is mapped to a single peer. The exact method of implementing such a mapping depends on the particular implementation of the FTT.

Example Figure 2 shows an example of a ring (a torus of dimensionality 1) and a mapping of ranges to peers. Peer p_1 is responsible for the range $(5, 10]$, p_2 is responsible for $(10, 15]$, p_3 is responsible for $(15, 18]$, p_4 is responsible for $(18, 20]$, and p_5 is responsible for $(20, 5]$. As can be seen, each region of the ring space is mapped to one (and only one) peer. Now, assume that peer p_1 fails or leaves the system. In this case, the FTT needs to reassign the range $(5, 10]$ to another peer. If a peer can be responsible for only one region of the space (one range), then p_2 or p_5 need to increase their range by taking over p_1's range.

2.2 Data Store

The Data Store is responsible for distributing the data items to peers. Ideally, each peer should store about the same number of items, achieving storage balance. The Data Store maps each data item to a point in the torus space, and it stores the item at the peer responsible for the region containing that point. If a peer stores many more data items than other peers, the Data Store will re-balance by splitting the region (and the data items) of the heavily loaded peer and assigning part of this region (and the corresponding items) to another peer. Exactly how this splitting is done depends on the specific instantiation of this component.

Example Looking again at Figure 2, assume that a data item t_1 mapped to value 6 is inserted into the system. The pair $(6, t_1)$ will be stored at peer p_1 as shown in Figure 3.

2.3 Replication Manager

The FTT 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

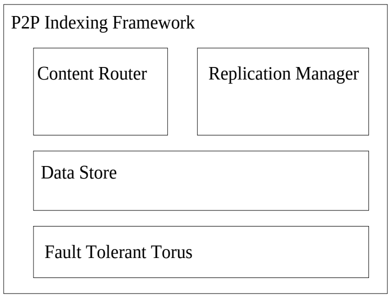

Figure 1: P2P Indexing Framework

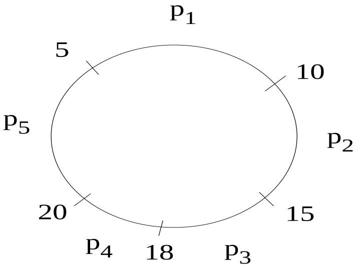

Figure 2: Mapping Ranges to Peers

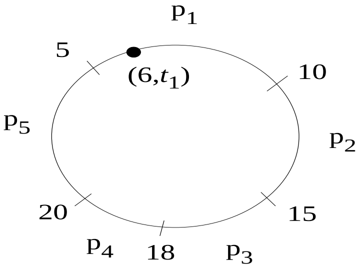

Figure 3: Mapping Values to Peers

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 P2P system are reliably (under reasonable failure assumptions) stored at some peer in the system until the items are explicitly deleted.

Example In Figure 3, if peer p_1 fails, peer p_2 or p_5 will take over the range $(5, 10]$ (as ensured by the Fault Tolerant Ring component). However, the data item $(6, t1)$ would be lost (6 is the index value of data item $t1$). However, by replicating the data item $t1$ at another peer in the system, the data item can be recovered, even if peer p_1 failed.

2.4 Content Router

The Content Router is responsible for efficiently routing messages to their destination in the P2P system. This is the component that supports the index search primitives.

3. INSTANTIATING INDEX STRUCTURES

Chord [6] can be instantiated in our framework as follows. The FTT is implemented using Chord's fault-tolerant ring, and each peer is assigned an ID on the ring using consistent hashing. The Data Store is implemented using a hash based scheme that hashes data items to values on the ring, and assigns the item to the first peer whose ID appears immediately after the value in the ring. The Replication Manager is instantiated using the techniques proposed in CFS [3]. The Content Router is implemented using Chord's finger tables.

P-Ring [2] is our novel P2P index structure. In devising the P-Ring, we were able to reuse the fault-tolerant torus of Chord and its replication mechanism because of our modularization. In addition, P-Ring has a new Data Store that is capable of partitioning and storing contiguous ranges of items in the peers, such that each peer is responsible for approximately the same number of the data items even if the underlying data distribution is heavily skewed. Unlike the Chord Data Store, the P-Ring Data Store does not hash data values, and thereby preserves the data ordering that is required for range queries. P-Ring also has a new Content Router that can provide logarithmic search performance even under highly skewed data distributions.

Skip Graphs [1] are designed to handle range queries. However, Skip Graphs as proposed can only handle one data item per peer. Using our modularized framework, we extend Skip Graphs to support multiple items per peer by reusing the Data Store from P-Ring. Skip Graphs as proposed fit in the Content Router component of our architecture.

4. DEMONSTRATION OVERVIEW

We have a running implementation of all three of the above index structures in a distributed environment using C++. Our demonstration will be run on up to 5 distributed computers in the demo room, and each computer will host up to 10 "virtual peers" (since we wish to show up to a total of 50 peers). We will provide a visualization window that will allow users to see the current state of the components of the different virtual peers and to issue search queries.

We propose to show the following aspects in the demo.

- Search queries: Users can issue search queries to distributed peers, and visualize the results on the visualization window. The searches can be equality and/or range queries, depending on the index structure.
- Monitor system: Users can use the visualization window to see the current status of the various components of the system.
- Introduce failures: Users can choose to kill arbitrary peers in the system; a visualization on each peer shows how the system recovers from the failure.
- Add peers to the system: Users can choose to add peers to the system, and a visualization at each peer shows how the index structure incorporates the new peer.
- Quantify component costs: Users will be presented with statistics in terms of message overhead for the various system components. This will quantitatively illustrate the costs of the different components.
- Compare index structures: Since all index structures will be running concurrently in a similar environment, users can monitor the overheads of the different index structures, broken down by each component. This will help illustrate the benefits of the different index structures under stable conditions and during failures.

5. REFERENCES

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

[2] A. Crainiceanu et al. An indexing framework for peer-to-peer systems. Submitted for publication, 2004.

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

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

[5] L. Galanis et al. Locating data sources in large distributed systems. In *VLDB*, 2003.

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