A Case for Locally-Organized Peer-to-Peer Lookup Services

Prasanna Ganesan Qixiang Sun Hector Garcia-Molina
Computer Science Department
Stanford University, Stanford, CA 94305, USA
{prasannag, qsun, hector}@cs.stanford.edu

Abstract

Distributed lookup services have predominantly fallen into one of two categories: Gnutella-based systems and DHTs. In this paper, we identify a set of applications for P2P lookup services, and analyze each of their requirements along a commonly-chosen set of dimensions. We show that neither Gnutella nor DHTs may provide the desired trade-offs among these dimensions. We go on to demonstrate a locally-organized P2P lookup service that matches more closely with these application requirements.

1 Introduction

The last few years have seen the emergence of distributed lookup services as perhaps the most popular application on peer-to-peer systems. A lookup service is defined to be one that maintains a dynamic set of key-value associations, and permits queries that request values associated with keys. Existing lookup services fall into one of two broad categories: Gnutella-style systems and Distributed hash tables (DHTs).

Systems of the first kind, including Gnutella[2] and its variants, do not attempt to organize the content in the network. If a peer node has some $\langle key, value\rangle$ pairs it wants to "insert", it simply stores them itself. Consequently, a query is answered simply by flooding the network with the query and gathering the results obtained at each node. In contrast, DHTs, such as Chord[6] and CAN[5], organize the network nodes into a specific topology and carefully place $\langle key, value\rangle$ pairs in the network to enable efficient retrieval of query answers.

Both approaches have their own advantages. Gnutella provides a robust system that adapts well to highly dynamic content and to nodes constantly joining and leaving the network. The strength of DHTs is their highly efficient search, and they are ideal for searching rare keys that have few values. Such searches tend to be expensive on Gnutella where one would have to flood the entire network.

In this paper, we advocate the need for a new scheme, "in between" Gnutella and DHTs, to provide support for applications that have an inherent concept of locality. We begin by examining some (potential) real world applications and their requirements in Section 2. We then look at how well Gnutella and DHTs satisfy these requirements in Section 3. In Section 4, we provide a brief summary of one such "in-between" scheme — the YAPPERS system [3] — to illustrate our vision of locally-organized P2P lookup services.

2 Applications of P2P Lookup

We analyze our proposed applications with respect to their requirements along the following axes:

- *Querying power.* The set of keys used to specify a search query could vary widely, from a single key specifying a restrictive "point" query, to a complex regular expression requiring all values whose keys match the regular expression.
- *External topology.* Many applications might require strict control over the peer-to-peer overlay network for reasons of security, access control, communication efficiency, or for establishing notions of locality.
- *Partial lookup v/s Total lookup.* Many applications do not desire all the values in the sys-

tem that match a given query. Typically, they would like to be presented with the k "best" answers for the query, which we call a *partial lookup*[1]. The notion of "best" is determined by the application, although many applications desire their search results to be "near" the source of the query in an externally-specified topology. Such a partial lookup is said to require *locality*.

- *Lookup Latency.* The latency requirements on queries can vary widely among applications, from tens of milliseconds to a few seconds.
- *Dynamism of Content.* The rate at which content is added to, removed from, or modified in the lookup service influences the trade-offs to be made between query and update costs in a lookup service.
- *Dynamism of Nodes.* The rate at which nodes themselves join and leave the network is another important factor. If a node holds $\langle key, value \rangle$ pairs, these have to be imported from or migrated to other nodes during node joins and leaves respectively. Non-graceful failure of nodes would also have to be taken care of. In addition, the system would have to maintain a consistent state under dynamic conditions.
- *Query Efficiency.* Finally, all applications desire as efficient a lookup solution as possible, although various measures of efficiency are conceivable. We will consider the total bandwidth consumption per query as a measure of efficiency.

Table 1 summarizes the requirements of the different applications along these axes as we now proceed to describe the applications in detail.

2.1 File Sharing

File-sharing systems are arguably the most popular application of peer-to-peer technology. The barrier to using a centralized solution has been more legal than technological, as evidenced by the superior performance of Napster, as compared to Gnutella. However, P2P solutions are also desirable when there is no clear business incentive to invest in a central server, when the shared content is dynamic, or if nodes frequently move in and out of the system.

In terms of querying power, clients would require fuzzy search to deal with misspellings, as also allow for more structured schemas on the key. For example, an MP3 music file could have multiple attributes such as the song title, artist, album, genre and year, and clients could pose conjunctive queries on some subset of the attributes. A client eventually wants only a few locations of the same file for a point query. However, these few locations should be the "best" for that client in terms of network latency or bandwidth.

2.2 Local Directory Services

With the increasing focus on wireless roaming internet access, the vision of a future where one roams around a city with a handheld device and connect to nearby computers will soon be a reality. In such a setting, an important application area is the use of P2P for providing local directory information. Businesses, such as restaurants, retail stores, or movie theatres, in an area could run their own computer as part of a global P2P network and "publish" relevant information. Thus someone walking by with a wireless device can pose queries of the form "Find all the Italian restaurants in this area with a current waiting time less than 10 minutes", or "find all shops in the area selling kites under $50."

Locality is critical. A client's physical location is an inherent part of every query, and the desired subset of answers is required to be physically near the source of the query. Though the meta-information associated with content (e.g. business items) would be fairly stable, the content itself may change very frequently. As an example, the movies being screened at a theater would not change rapidly, but the number of available seats for a particular showing could. Directories are naturally well-structured and hierarchical, and queries can be treated as simple point queries, or queries over all keys within a directory subtree.

2.3 P2P Cooperative Caching

P2P caching has been proposed as a means of relieving a web server of some of its burden, especially when the server is confronted by flash crowds

[1] We note that we could conceivably alter the definition of the key in order to convert partial lookups into total lookups, for example, by somehow ensuring that only the "best" answers have the same key, but such manipulation is not always possible.

	File Sharing	Local Directory Service	Cooperative Caching	Referral Systems
Partial lookup with locality	$\surd$	$\surd$	$\surd$	
External Topology				$\surd$
Querying Power	fuzzy	point/range	point	
Lookup Latency	$O(1s)$	$O(1s)$	$O(0.01s)$	
Content Dynamism	Low	High	High	
Node Dynamism	High	Very Low	High	Low

Table 1: Summary of application requirements.

[4]. The idea is that a client in need of a piece of content contacts other clients "near" it which already have the content, and downloads from one of them, instead of requesting the content from the web server. In this fashion, we reap the benefit of a virtual cache containing all the content that all the clients have downloaded, while keeping the solution much more cost-effective than setting up real proxy caches across all clients.

Locality is important as a client needs to identify the "best" cached copy in terms of latency and bandwidth. The process of finding a cached copy or deciding that no cached copy is available must be fast (on the order of tens of milliseconds). Otherwise, a user's web browsing session would be excruciatingly slow. Point queries are the norm as web URLs form a unique identifier. For a given URL, the content may change frequently, e.g., headline news or ads. The lookup service should thus be able to invalidate old versions and register new versions efficiently.

2.4 Social Networks

A fertile but nascent area for P2P applications are Social Networks where every peer node is connected directly to all of their friends. Simple applications of social networks include inviting all friends and friends of friends to a party [1], or a referral system where a peer looks for experts on a subject as few degrees removed from it as possible. Social networks can also be the basis of building reputation-based systems where each peer only connects to other peers it trusts. Such construction can easily control or minimize the impact of malicious behaviour. Clearly, social networks have one dominating property: external topology. The connection topology is given or dictated by other criteria. A lookup service built on top of a social network *cannot* alter the topology at will. In Table 1, we demonstrate the properties necessary for a referral system. A reputation-based file-sharing system would demonstrate the same requirements as a normal file-sharing system, together with the need for complete control over the topology.

3 Why not Gnutella or DHTs?

With the properties listed in the previous section, we now argue that neither Gnutella nor DHTs match the requirements of the applications specified earlier. Table 2 offers a letter grade to indicate how well Gnutella or DHTs match the various application requirements we have outlined, with the final column indicating the ideal trade-off that the application would like to make.

Gnutella can provide very flexible queries, easy adaptability to partial lookups with locality, and coexistence with arbitary external topology. For example, in order to implement local directory services, we would want to ensure that nodes that are physically close are also close in the overlay. Gnutella also adapts well to content and node dynamism since it maintains very little distributed state. However, these systems are very inefficient because of the bandwidth and messaging overhead associated with each query. This inefficiency can prove to be severe enough to prevent the overall system from scaling up to a large number of nodes and render it impractical or expensive.

In contrast, DHTs offer efficient lookup at the expense of the other properties. Point queries are the main kind of queries supported, although range queries can also be supported with acceptable loss in performance. In DHTs such as Chord, the only

	Gnutella	DHTs	Application Requirement
Partial lookup with locality	A	B-C	A
External Topology	A	F (Chord) C (CAN)	A
Querying Power	A	C	B
Lookup Latency			
Total Lookup	C	A	C
Partial Lookup	B	B	A
Content Dynamism	A	B	B
Node Dynamism	A	C	B
Query Efficiency	D	A	B-C

Table 2: Application Requirements and the Quality of Solutions offered.

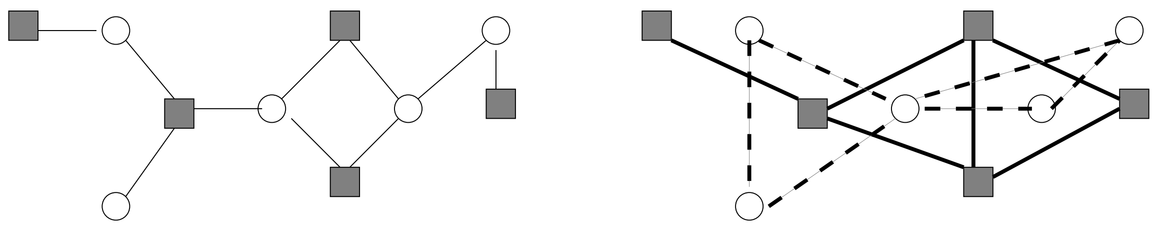
way of supporting an external topology is to treat the external topology as a "network layer" and build an additional overlay on top of this layer to implement DHTs. Such an additional layer can, by itself, push the query latency to unacceptable lengths. CAN can support external topologies a little better since all of a node's neighbors are near the node in the hash space. However, supporting an arbitrary external topology is still hard and may lead to poor and unbalanced CAN configurations.

Equally problematic is the need to capture locality and provide partial lookup in a DHT. One way to do so would be to ignore the locality requirement for lookup, and simply require the originator of the query to process all the returned answers and filter out those answers that are not necessary. Such a methodology has many drawbacks; we fail to exploit the fact that not all anwers need to be produced for a query, leave ourselves open to hot-spot issues if all the answers are stored at a single node, and do not have locality of access since relevant index entries are not stored close to the nodes which require them. An alternative solution would be to offer coarse-grained local search by using some kind of locality information, such as a zip code, as part of the key, but such solutions do not work for all applications. Although DHT lookups are efficient, query latency can still be rather high in absolute terms. Performance overhead in dynamic conditions is worse than in Gnutella for two reasons. One, more distributed state must be maintained, and such state may be shared between nodes arbitrarily far away in the physical network. Two, data migration is necessary as nodes join and leave in order to keep the content in the system and keep it balanced across the nodes. Furthermore, dealing with non-graceful failures requires even more overhead. Gnutella nodes are responsible for only their own content, and taking their content offline along with them is acceptable for many applications.

We claim that what these applications really need is a different set of trade-offs, as illustrated in the right-most column of Table 2. Applications are willing to sacrifice the ability to perform completely arbitrary queries, and to maintain some amount of distributed state, in exchange for good support for external topologies, and low-latency partial-lookups with locality. We now demonstrate a solution that addresses these requirements.

4 An Alternative: YAPPERS

YAPPERS (Yet Another Peer-to-PeER System) is a first attempt at building an "in-between" scheme that tries to combine Gnutella-style flooding with DHTs. Conceptually, YAPPERS takes any arbitary overlay as input. It then partitions the nodes into a fixed number of buckets by applying a hash function to each node's IP address. Figure 1(a) gives a simple example where YAPPERS partitions the nodes into just two hash buckets: circles and squares. Once the nodes have been divided, YAPPERS forwards search queries between two nodes of the same hash bucket if these nodes are near each other in the original overlay. Figure 1(b) illustrates how these forwardings induce two search networks: one circle network and one square network. In effect, YAPPERS creates a fixed number of smaller Gnutella networks that re-



(a) Raw overlay **(b) Induced search networks**

Figure 1: YAPPERS divide nodes in the raw overlay into two buckets which induce two search networks.

spects the node proxmities in the original overlay.

To perform a $\langle key, value\rangle$ pair insertion or a query for a particular key, a node simply contacts the closest node that is assigned the corresponding hash bucket. YAPPERS, in constructing the smaller search networks, actually ensures that there exists at least one member of each search network within a certain number of hops of each node. For example in Figure 1, each node can find a circle or square node within one hop. Thus from each node's perspective, all operations are essentially performed on a small DHT that consists of nodes within a certain number of hops. Details can be found in [3].

5 Analysis and Conclusion

YAPPERS, being a locally-organized network built on top of Gnutella, brings with it Gnutella's advantages, including the ability to work with arbitrary topologies, to optimize for partial lookups and exploit locality of reference. Since content is organized locally and leads up to a global search structure, we have low-latency partial-lookups, with the querying power limited only by the granularity at which the buckets are built. When content and nodes are dynamic, the amount of work involved in maintaining a consistent state is higher than in Gnutella but is still manageable especially given the fact that all such interactions required are purely local. Data migration issues on node failures can be handled cleanly by requiring the "owner" of content to ensure its availability. This scheme is efficient since the owner is close to the location where the content is stored and will automatically become aware of any node failures.

In conclusion, we claim that any application with an external topology, with locality requirements on answers, or requiring low-latency partial-lookups will benefit from using locally-organized lookup services. Locally-organized services provide exactly the right trade-offs required by these applications while often providing a more efficient solution than plain Gnutella.

References

[1] Club Nexus. http://clubnexus.stanford.edu/.

[2] Gnutella. Website http://gnutella.wego.com.

[3] P. Ganesan, Q. Sun, and H. Garcia-Molina. Yappers: A peer-to-peer lookup service over arbitary topology. To appear in *INFOCOM* 2003.

[4] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *NOSSDAV*, 2002.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, August 2001.

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 160–177, San Diego, August 2001.