PEER-TO-PEER DISTRIBUTED LOOKUP SERVICES

ABSTRACT

As we are living in the era of technology innovation, the need of sharing information is becoming more important than ever. Peer-to-peer applications were introduced and developed in order to facilitate particularly the sharing of large video/audio files and software. Although the entire peer-to-peer research literature consists of four divisions: search, security, storage, and applications, in this paper, I will focus on the first division; search method or Peer-to-peer (P2P) distributed lookup service which is one of the most popular Internet applications.

There are two types of algorithm in the P2P routing: “structured” and “unstructured”. Gnutella is the typical application for the unstructured approaches in which keywords queries are widely flooded. On the other hand, the structured algorithms which support *distributed hash table* (DHT) will guarantee location for a target within a determined number of hops. Although DHT did repair some scaling problems in the unstructured approach and has been referred to as the “second generation”, it is inherently ill-suited to range queries which are more prevalent and important in practice. Also, DHT will become inefficient when peers are continually arriving and departing. Therefore, I assert that it is more worthy to retain the simplicity of Gnutella while proposing new mechanisms to improve its scalability than to build a truly scalable DHT.

1. INTRODUCTION

Unstructured P2P routing, Gnutella, does not attempt to organize the content in the network. A peer node will store the (key, value) pairs it has by itself. Therefore, a query is answered simply by flooding the network with the query and gathering the results obtained at each node. In contrast, DHT-based systems, such as Tapestry, Pastry, Chord, and Content-Addressable Networks (CAN), organize the network nodes into a specific topology and carefully place (key, value) pairs in the network to enable efficient retrieval of query answers. The hash-table-like lookup() operation provided by DHT typically requires only $O(\log n)$ steps, whereas in comparison, Gnutella requires $O(n)$ steps to reliably locate a specific file.

The DHT did provide a scalable replacement for un-scalable Gnutella-

like file sharing systems. There is no denying this; but DHT also has inherent weaknesses including high overhead costs, limited lookup method, and lack of mass-search ability.

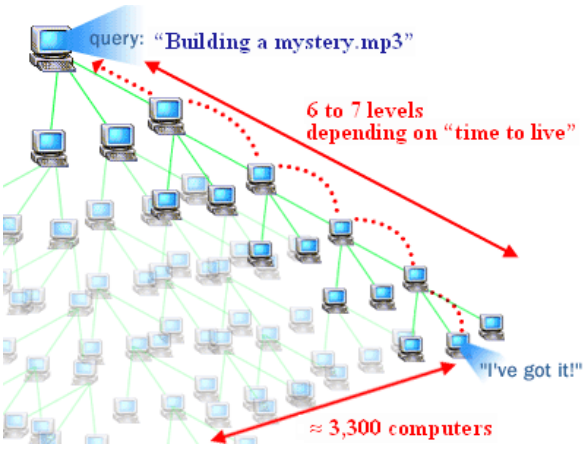
I will begin Section 2 with a brief overview of Gnutella and DHT-based algorithms technology. This is followed by the discussion on the sizeable obstacles of the deployment of DHT system, and why, designers will have a difficult time seeing the deployment of a DHT solution. In Section 4, I attempt to dispute the main counter-argument about the scalability of Gnutella-based system. Recent improvement in Gnutella will be discussed in Section 5.

2. REVIEW OF GNUTELLA AND DHT TECHNOLOGY

2.1. Gnutella technology

To begin with, I will give a brief overview of Gnutella's file sharing methodology. As I mentioned, in Gnutella network, there is no central server to store the names and locations of all available files. When a peer node raises a query, such as name of a song, Gnutella will identify the location of requester by its IP address. Then the query will be sent to other Gnutella peer nodes on the network. These machines search to see if the requested file is on the local hard disk. If so, they send back the file name (and machine IP address) to the requester. Simultaneously, all of these machines send out the same request to the machines they are connected to, and the process repeats. In order to limit to numbers of machines involving in each research, each request has a Time to live (TTL) limit placed on it. Usually, a request might go out six or seven levels deep before it stops propagating.

Suppose each machine on the Gnutella network knows three other counterparts, with six level depth, a query will reach approximately 3,300 ($3+3^2+3^3+3^4+3^5+3^6$) other machines.



Another important thing is that all queries in Gnutella are kept separate. Each query has a unique number, generated from random numbers or semi-randomly from something unique to the requester. This enables a peer node to recognize a repeat query and quietly drop it. On the other hand, different machines can request the same

material and have their requests satisfied because each has a unique identifier.

Gnutella, therefore, can distribute a query to thousands of machines quickly in the simplest way. It *works all the time* as all peer nodes are dependent to each other.

2.2. DHT technology

This section will continue with the review of existing DHT algorithms. The main principal is that all of them take, as input, a `key` and, in response, route a message to the node responsible for that key. When a node receives a query for a key for which it is not responsible, the node routes the query to the neighbor node that makes the most "progress" towards resolving the query. Depending on the algorithms, the notion of progress will be different, but in general is defined in terms of some distance between the identifier of the current node and the identifier of the queried key.

Plaxton tree: Plaxton tree is perhaps the first structured algorithm that could be used by DHT. It influences the designs of Pastry, Tapestry, and Chord. This routing algorithm works by "correcting" a single digit at a time: if node number 36278 received a lookup query with key 36912, which matches the first two digits, then the routing algorithm forwards the query to a node which matches the first three digits (*e.g.*, node 36955). For a system of n nodes, each node has O(logn) neighbors. Since one digit is corrected each time the query is forwarded, the routing path is at most O(logn) overlay hops.

Assuming a relative static node population, Plaxton tree can locate objects using fix-length routing tables. If the latency matrix whose size is n^2 is known, by using the shortest-path algorithm, the routing tables can be chosen to minimize the expected path latency between two nodes.

Tapestry: Tapestry is a variant of Plaxton tree algorithm. The algorithm maintains the properties of having O(logn) neighbors and routing with path lengths of O(logn) hops.

Pastry: Pastry uses Plaxton-like prefix routing. Each node maintains a "leaf set" and a "routing table". The leaf set contains L/2 node IDs which are smaller than local node ID, and L/2 node IDs which are larger than the local node ID (that is, half larger, half smaller). Correct routing can be achieved with this leaf set. The routing table helps achieve more efficient routing. The methodology to use the routing table is not described here. Routing consists of forwarding the query to the neighboring node that has the longest shared prefix with the key (as in Plaxton tree algorithm). Pastry has O(logn) neighbors and routes within O(logn) hops.

Chord: Chord uses a one-dimensional circular key space. The node responsible for the key is the node whose identifier most closely follows the key; that node is called the key's *successor*. Chord maintains two sets of neighbors at each node: *successor list* and *finger table*. The s*uccessor list* which has k nodes immediately follows the node's key in the key space. Routing correctness is achieved with this *successor list*. Routing efficiency is achieved with the *finger tablet* of O(logn) nodes spaced exponentially around the key space. Routing consists of forwarding to the node closest the key; path lengths are O(logn) hops.

CAN: CAN chooses its keys from a d-dimensional Cartesian coordinate space. Each node is responsible for a hypercubal region in this coordinate space, and its neighbors are the nodes that "own" the contiguous hypercubes. Routing consists of forwarding to a neighbor that is closer to the key. Nodes have O(d) neighbors and path lengths are $O(dn^{1/d})$ hops. When d=logn, CAN has O(logn) neighbors and O(logn) path lengths like the other algorithms.

3. WEAKNESSES OF DHT-BASED SYSTEMS

3.1. High overhead costs

P2P clients are extremely transient. Measured activity in Gnutella indicates that the median up-time for a node is 60 minutes. For large systems of, for example, 100,000 nodes, this implies a churn rate of over 1,600 nodes coming and going per minute. Churn does result in graceless failures for DHT, which in turn, requires significant overhead to maintain the network.

Graceless failures occur when a node fails without informing its neighbors and transferring the relevant state. These failures require more time and work in DHT to (a) discover the failure and (b) re-replicate the lost data or pointers. In order to preserve the efficiency and correctness of routing, most DHT requires O(logn) repair operations after each failure. If the churn rate is too high, the overhead caused by these repair operations can become substantial and could easily overwhelm nodes with low-bandwidth dial-up connections.

However, churn causes little problem for Gnutella as long as a peer doesn't become disconnected by the loss of all of its neighbors, and even in that case

the peer can merely repeat the bootstrap procedure to re-join the network.

3.2. Limited lookup method

DHT organizes the network nodes into a specific topology, carefully places (key, value) pairs in the network, and uses exact-match key researches. When a query is received, DHT will translate the exact name of the requested file into a *key* and perform the corresponding lookup(key) operation. This method is perfect for exact-match queries. However, DHT has enormous difficulties in supporting *keyword* searches, i.e. finding files that match a sequence of keywords.

For example, we have a request to find “Amazing Grace” song of which Leann Rimes is the singer. Users will submit a search in the form “Amazing Grace Leann Rimes” and expect the system to search for files that match all of the keywords in the search query. However, this is a non-trivial task to DHT. I would like to emphasize that there is no clear naming convention for file names in the P2P systems. Thus, the same piece of content is usually stored by different nodes under several slightly different names which make it almost impossible for the DHT to match the key with the values. Some may argue that DHT can overcome this issue by constructing an inverted index per keyword. However, I think this approach will be complicated because it requires additional caching algorithms needed to avoid overloading nodes that store the index for popular keywords. In addition, higher maintenance cost will be non-evitable, especially when we consider the node churn issue.

In contrast, Gnutella effortlessly supports keyword searches since all such searches are executed locally on a node-by-node basis. The important thing is that in reality, keyword searches dominate in the quantity as well as the importance to exact-match queries.

In addition, if P2P networks are going to be adaptable, and they are to support a wide range of applications, then they need to accommodate many search types, albeit with relaxed semantics. Therefore, that DHT is inherently ill-suited to range queries is a big barrier for its further development. According to Bharambe, Agrawal et al. 2004 [14], the very feature that makes for their good balancing properties, randomized hash function, works against range queries. One possible solution would be to hash ranges, but this requires a priority partitioning. If the partitions are too large, partitions risk overload. If they are too small, there may be too many hops.

3.3. Lack of mass-search ability

According to Chawathe, Ratnasamy et al. 2003 [1], if a file is requested frequently, it will be cached in many places on the network and is referred to as *hay*. In contrast, file which has only a single copy in the system is known as *needles*. Consequently, as long as given the exact name of a file, DHT with exact-match queries would allow users to find it, even though there is only one copy in the system. By applying the time to live for each request, Gnutella file-sharing system is able to limit the query propagation. However, Gnutella cannot guarantee to find the requested file in the number of nodes which a query can reach. In other words, DHT is superior in finding "needles", whereas Gnutella has advantage in searching "hay".

In the current file-sharing deployments, searching for "hay" is more prevalent. John L. Hennessy and David A. Patterson state "In making a design trade-off, favor the frequent case over the infrequent case so that to make the common case fast" in the textbook "Computer Architecture – A quantitative Approach 3rd Edition" page 39. I do not intend to refuse the importance of DHT as a searching application. However, it serves only for a minority of queries. Considering putting effort on improving the scalability of either DHT or Gnutella, I believe it will be more efficient and cost-saving if we focus on Gnutella.

4. COUNTER ARGUMENT

The main counterargument against the unstructured algorithms of which Gnutella is the representative focuses on the scalability. Critics argue that Gnutella is inherently not scalable, and thus should be abandoned. There is no denying that scalability is the critical problem of Gnutella; but it does not mean that this file-sharing system is useless.

First, Gnutella does support well keyword searches as well as well-replicated file lookups which are the majority. Second, many recent researches reported positive result of improving the Gnutella's scalability. Recent improvement in the Gnutella's techniques will be summarized in the next part.

Gnutella's scalability issues could be improved by making better use of the more powerful peers. Instead of flooding mechanism, random walks are used. Many recent researches focus on a bias for high degree peers and very short directed query paths and concern load on the "better" peers. For example, Chawathe, Ranasamy et al. 2003 [1] incorporated several new features to the early Gnutella designs. They devised a topology adaptation algorithm so that most peers are attached to high-degree peers. They used random walk search algorithm and bias the query load towards high-degree peers. For one-hop replication, they required all nodes keep pointers to content on adjacent peers. A

peer must have signal from its neighboring peer before sending a query to this neighbor. The result was scalability improvement of three to five orders of magnitude over Gnutella while still retaining the robustness.

5. IMPROVEMENT IN THE GNUTELLA TECHNIQUES

Gnutella Forum had numerous enhancements for the unstructured file-sharing system. One of the main improvements is that an index of filename keywords, called the Query Routing Table (QRT), can now be forwarded from 'leaf peers' to it 'ultrapeers' [10]. Ultrapeers could then ensure that the leaves only receive queries for which they have a match. This results in dramatically reducing the query traffic at the leaves.

More recently, there has been a proposal to distributed aggregated QRTs among ultrapeers [11]. To further limit traffic, QRTs are compressed by hashing, according to the Query Routing Protocol (QRP) specification [10]. This same specification claims QRP may reduce Gnutella traffic by orders of magnitude, but cautions that simulation is required before mass deployment. Similarly, Gnutella UDP Extension for Scalable Searches (GUESS) [12] aims to reduce the number of queries for widely distributed files.

The broader research community has recently been leveraging aspects of the Gnutella design. Chawathe, Ratnasamy et al. 2003 [1] argued that by making better use of the more powerful peers, Gnutella's scalability issues could be alleviated. They introduced a new technique called random walk to replace the Gnutella's flooding mechanism. They claimed that there was scalability improvement of three to five orders of magnitude over Gnutella while still retaining the robustness.

Castro, Costa and Rowstron [13] argued that if Gnutella were built on top of structured overlay, then both the query and overlay maintenance traffic could be reduced.

In summary, these improvements in Gnutella are characterized by a bias for high degree peers and very short directed query paths, and concern about excessive load on 'better' peers.

6. CONCLUSION

I believe that DHT, while more efficient at many tasks, is not well suited for mass-market file sharing. In particular, its ability to find exceedingly rare files is not needed in a mass market file-sharing environment, while their ability to efficiently implement keyword search, which is crucial for this application, is still unproven. During a churn, the maintenance cost of the DHT-based system will increase significantly,

while its reliability is not guaranteed. Although still struggling with scalable issues, Gnutella has more query power, thus supports the mass-market file sharing better. Moreover, it is evident that improvement of Gnutella's scalability is feasible. Gnutella, with the above mentioned strengths, deserves to get more effort to develop than DHT-based approach.

References:

[1] Chawathe, Y., S. Ratnasamy, L. Breslau, N. Lanham and S. Shenker (2003). "Making gnutella-like P2P system scalable." 2003 conference on Applications, Technologies, Architecture and Protocols for Computer Communication, Karlsruhe, Germany, August 25-29, 2003

[2] J. Risson and T. Moors. "Survey of Research towards Robust Peer-to-Peer Networks: Search Methods." Technical Report UNSW-EE-P2P-1-1, University of New South Wales, Sydney, Australia, (September 2004)

[3] S. Ratnasamy, S. Shenker and I. Stoica. "Routing algorithms for DHTs: Some open questions." *1st Workshop Peer-to-Peer Systems (IPTPS'01),* Cambridge, MA, 2001

[4] Plaxton, C., Rajaraman, R., and Richa, A. "Accessing nearby copies of replicated objects in a distributed environment." *ACM SPAA* (Newport, Rhode Island, June 1997)

[5] Zhao, B. Y., Kublatowicz, J., and Joseph, A. "Tapestry: An infrastructure for fault-tolerant wide-area location and routing." Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Department, 2001.

[6] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. "Chord: A scalable peer-to-peer lookup service for internet applications." *ACM SIGCOMM '01 Conference* (San Diego, California, August 2001)

[7] Druschel, P., and Rowstron, A. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems." *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)W* (Nov 2001)

[8] Ratsanamy, S., P. Francis, M. Handley, R. Karp and S. Shenker (2001). "A Scalable content-address network." ACM SIGCOMM 2001

[9] http://computer.howstuffworks.com/file-sharing3.htm

[10] Singla, A. and C. Rohr (2002). "Ultrapeers: Another Step Towards Gnutella Scalability"

[11] Fisk, A. (2003) "Gnutella Ultra Peer Query Routing"

[12] Daswani, S and A. Fisk (2002). "Gnutella UDP Extension for Scalable Searches (GUESS) v0.1."

[13] Castro, M., M. Costa and A. Rowstron (2004). "Should we build Gnutella on a structured overlay?" ACM SIGCOMM Computer Communication Review

[14] Bharambe, A., M. Agrawal and S. Seshan (2004). "Mercury: Supporting Scalable Multi-Attribute Range Queries." SIGCOMM'04, Portland, Oregon, USA, August 30 – September 3