Distributed Hash Tables in P2P Systems - A literary survey

Timo Tanner
Helsinki University of Technology
tstanner@cc.hut.fi

Abstract

Distributed Hash Tables (DHT) are algorithms used in modern peer-to-peer applications, which provide a reliable, scalable, fault tolerant and efficient way to manage P2P networks in a true peer to peer manner. A lot of literature has been published on the subject to analyze various different DHT algorithms, their properties and performance.

The purpose of this paper is to find out about recent research conducted on the subject and wrap up a literary survey containing a comparison of four different DHT algorithms. The algorithms chosen for the survey are Chord, Tapestry, CAN and Kademlia. The comparison concentrates on the relative efficiencies of the different DHTs, presents them and analyzes the factors that affect the efficiency of a Distributed Hash Table.

KEYWORDS: Distributed Hash Table,Network Overlay,Distance function,Node,Node state,Churn,Number of hops,Latency,Network partition

1 Introduction

Hash tables are data structures that can be used to map keys to values, store key/value pairs, and retrieve values using the given keys. They are very useful and efficient in traditional applications. A need for a similar solution in distributed computing has sprung up, and in recent years, a lot of effort has been put into this research. It can be said that the heart of most P2P-systems is the search scheme that is used and that the first generation of P2P protocol solutions to this are not efficient, reliable and scalable enough. In the first generation P2P applications there are central servers that contain searchable information, which can become a bottleneck when the number of clients increases.

Distributed hash tables are similar to distributed data structures that are used in P2P applications to store and retrieve data efficiently. In a classic hash table, hashable objects are stored in different buckets according to each object's hash value which is obtained by applying a hash function to the data being stored. Since the information lookup mechanism is logarithmic, the systems that utilize DHTs are extremely scalable. When the number of nodes in the network doubles, only one more hop is needed to find any given node. Further, DHTs store the values on a number of hosts instead of just a single host.

Since P2P networks and applications function differently from traditional applications, centralizing critical data tends to create more problems than solve and thus, in DHT P2P networking, every node in the network has a Globally unique identifier. (GUID) Each node functions as a container for the distributed data.

Nodes in this type of P2P network are arranged to a Network overlay, which for example dictates how the nodes are connected and to how many neighbors each node can connect to. A network overlay is a virtual network that runs on top of another network and can be managed independently. A network overlay can utilize the underlying protocols as seen fit. Some DHTs prefer using connectionless UDP whereas others resort to the connection oriented TCP. In P2P world, the network overlay functions on the application level.

Distributed Hash Tables promote several ideas that distinguish them from traditional Client-Server oriented services:

1. DHTs provide decentralized operation, without the need to maintain a centralized server to control the P2P network. This also results in a system composed of many parts. In this sense, P2P applications using a DHT are true P2P applications.

2. The system is scalable, meaning that the system functions properly even with very large node count and traffic.

3. Load is balanced fairly among the peer nodes in the network, in a way that does not overencumber any node, which could occur int client and server model.

4. The system is based on the assumption that the network is not static and changes occur fairly frequently with nodes joining and leaving the network. This is called "Churn".

5. Routing and data retrieval is fast, and can be accomplished in logarithmic time.

6. The system is robust, meaning that it can withstand even large scale attacks.

1.1 Related work

There are several papers that introduce and analyze DHT algorithms, such as Tapestry [1], CAN [3], Chord [4, 14], Kademlia [7] and Pastry [5]. Each algorithm is introduced and analyzed with their respective strengths and weaknesses.

[15] discusses routing geometries in general. [6] analyzes routing and data lookup. [2] discusses Churn and [9] caching related things, such as comparison between proactive and reactive Churn recovery.

[8] addresses security problems in DHT P2P applications. [10, 11, 12] present performance studies of various algorithms with and without Churn. [13] introduces an O-notational performance analysis of P2P network overlays.

2 Introduction to different DHT algorithms

2.1 Chord

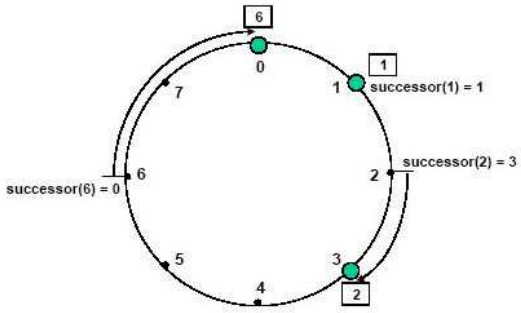

Figure 1: Chord has a ringlike routing geometry. The picture presents a Chord network that consists of 3 nodes that contain the values 6,1,2. The size of key space is three bits, leading to node IDs ranging from 0 to 7. [4]

Chord is a DHT algorithm that is based on key/value pairs. Topologically Chord resembles ring. In the network overlay, the id of the P2P network client represent the key and the corresponding value is the data that is stored at the corresponding nodes. In Chord, each node stores only a subset of all the keys, which increases the reliability of the network. Chord can be used to implement a number of different services, including distributed information lookup services. Chord has been designed with six goals in mind:

1. Scalability means that the system can still function efficiently when the size of the network grows.
2. Availability means that the system can function even when the network partitions or some nodes fail.
3. Load-balanced operation means that the key/value pairs are evenly distributed in the system.
4. Dynamism means that the system can handle rapid changes in network topology.
5. Updatability means that data can be updated using the DHT algorithm.
6. Locating according to "proximity" means that if search value is stored in a node near the query originating node, then the data should be fetched from there, instead of over great distances.

When the system is under high churn, it is possible for the overlay network to become partitioned. This happens when sufficiently many nodes leave the network simultaneously almost splitting the network in two separate subnets that have severely limited access to the other, at least in theory. Chord has been designed so that these kind of network partitions do not render the whole network unusable.

Chord also maintains a location table in each node. This table functions as a cache and contains the recent nodes that Chord has discovered while running. The cache is basically a routing table that maps node identifiers to their corresponding IP-addresses and ports. This cache can be used to speed up key lookups, when storing node info, topologically closer nodes are always preferred over farther ones. This is decided according to the round trip time to the target node.

On application level, Chord is divided into Chord server and Chord client, which both run on the same network node. Both ones are implemented as Remote Procedure Calls (RPC) and can handle over 10000 requests per second. [4]

No officially published Chord implementations exist at the moment of writing the paper.

2.2 Tapestry

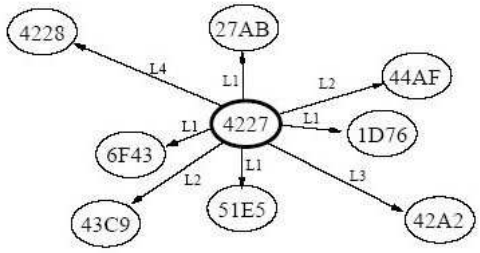

Figure 2: Tapestry routing mesh from the perspective of a single node. Outgoing neighbor links point to nodes with a common matching prefix. Higher-level entries match more digits. Together, these links form the local routing table. [1]

Tapestry is different from the other overlay network protocols in such a way that it considers the network distance when looking up keys. CAN and Chord do not do this but rely simply on hopcount, which can sometimes take the lookup to the other side of the network. Also, Tapestry allows nodes more freedom when publishing replicas of the routing information.

Tapestry uses neighbour maps and tree like routing to deliver the messages to target nodes. When a node sends a lookup, it transmits it to the addressses that are found in it's local neighbour map, which contains ID-number IP-address pairs. The node always forwards messages to those target nodes that are topologically closer to it.

Tapestry is a P2P overlay routing algorithm that can be used to implement distributed applications. Tapestry provides Decentralized Object Location and Routing in an efficient, scalable and location independent manner.

Oceanstore is a Java based Tapestry implementation that has been developed at UC Berkeley. Oceanstore is a distributed storage application that consists of untrusted servers. It can be downloaded freely from SourceForge. [16] [17]

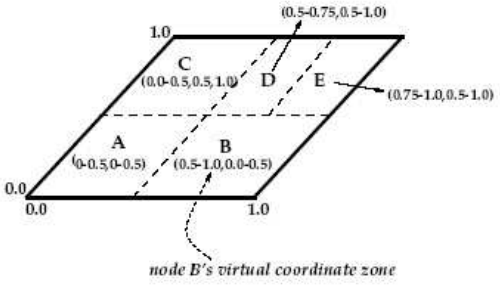

Figure 3: CAN network overlay. The alphabets denote coordinate space partitions and the number pairs the respective coordinate range of each partition. [3]

2.3 CAN

CAN stands for a Scalable Content Addressable Network. CAN's Topology is a d-dimensional Cartesian coordinate space. The coordinate space is a logical one and doesn't necessarily resemble any physical coordinate space. It is always entirely partitioned among all the nodes in the system, each node owning part of it. When a new node joins the network overlay, some reserved partition is split in half to allocate space for the new node. Node is another node's neighbour if their coordinate spaces meet along d - 1 axis. In 2-dimensional case, the partitions in coordinate space would have to meet along one axis.

Each node in the CAN system stores a part of the hash table and in addition also keeps information about the neighbour nodes that are closest to it. Closeness in this case is measured by hops.

CAN can be used to implement large scale storage management and retrieval systems. At least one prototype implementation of CAN is known to exist: pSearch is a semantic and content based search service that utilizes a hierarchial variant of CAN, eCAN.[18]

2.4 Kademlia

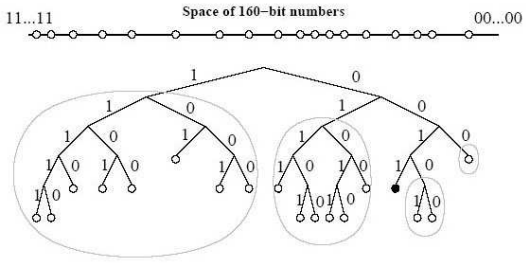

Figure 4: Kademlia routing tree. Each node in the binary tree represents a node in network overlay. [7]

When deciding where to store particular key/value pairs, Kademlia does a bitwise XOR operation on the 160-bit node id numbers and finds out the topological distance between them. Lookups are then forwarded to the nearest available node. Kademlia protocol states that each node must know at least one node in each of it's subtrees. This way every node can be located reached from any given node using it's ID.

Kademlia is a widely used algorithm that utilizes UDP protocol in communication. Kademlia algorithm has been implemented in eMule, mlDonkey, RevConnect DC++ and Overnet file sharing applications for indexed file searching. Forementioned P2P applications are not based on true peer networks since they all rely on hub or server oriented architecture that only use DHTs for file indexing.

3 Performance issues and evaluation metrics

3.1 Performance issues

3.1.1 Churn

In P2P applications, churn is a term that refers to the rate by which the network overlay changes, i.e. the rate of node arrivals and departures from the network. The manner applications handle these changes has a big impact on the system overall performance. It has been argued that many existing DHT systems cannot handle high Churn rates well. [2] Churn can be measured by the node "session time", which is the time from node join to node departure. The lower the average session time for network nodes, the higher the Churn. Node lifetime, on the other hand, is the time from first join to final departure.

3.1.2 Packet loss

Packet loss refers to the percentage of dropped packets from total amount sent. Packet loss is hard to estimate beforehand since it depends on the underlying network conditions. However, the DHT system should withstand occasional high packet loss rate without rendering the service unusable. In DHT utilizing systems, packet loss is often a symptom of network congestion.

3.1.3 Proximity routing

Proximity routing [6] is used in several DHTs to improve lookup times. When doing proximity routing, queries are directed to nodes that are geographically close to originating node to reduce the latency that is caused by message traversal in network. CAN, Kademlia, Tapestry and Chord all use Proximity routing in their latest versions.

3.1.4 Caching

Caching has been successfully used on various different applications and the same principle can be applied also to DHTs to improve their performance. There are basically two different caching schemes: passive [9] caching and proactive caching. [9] In passive caching, a copy of the requested node is stored in every node on the route from object storage node to querying node. In proactive caching, a small subset of the most popular resources are cached in every node beforehand. This can lower the average retrieval delay to 1 hop. It is argued that DNS, web accessibility and multimedia content distribution on the Internet could all benefit from proactive replication and caching of resources. [9]

3.2 Metrics

3.2.1 Number of hops

Number of hops or hopcount is an indicator that is used when measuring distances between nodes. A hop means one forwarding of a message from one network node to another.

3.2.2 Latency

Latency is an indicator that is used when measuring key lookup delay. Latency is usually the same as the round trip time for a query to come back to the originating node.

3.2.3 Stabilization interval

Some P2P systems control the network overlay using stabilization. Stabilization means removing routing data for nodes that have either left the network or dropped out. Stabilization interval indicates how often this is done.

4 Performance results

4.1 DHT performance

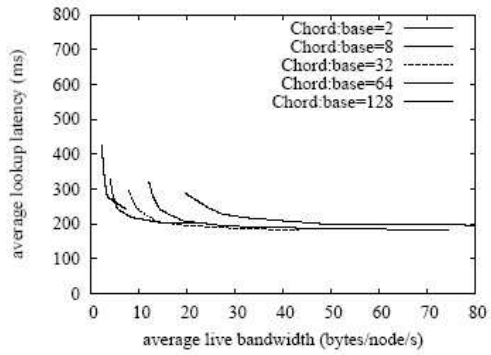

Figure 5: Lookup latencies for Chord. Base b is a modifier that dictates the number of items in routing tables. Number of entries in single node's routing table equals (b-1) log_b (n)

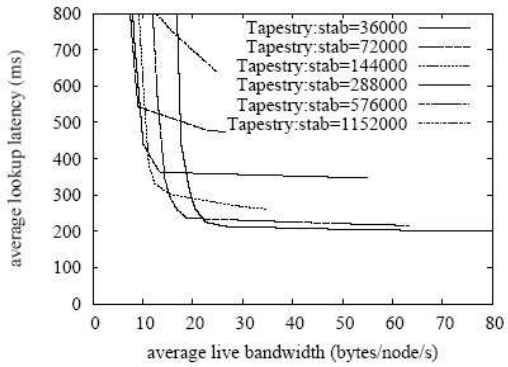

Figure 6: Tapestry lookup latencies for different stabilization times. (in milliseconds) Y-axis: lookup latency in milliseconds, x-axis: traffic/node/s. [12]

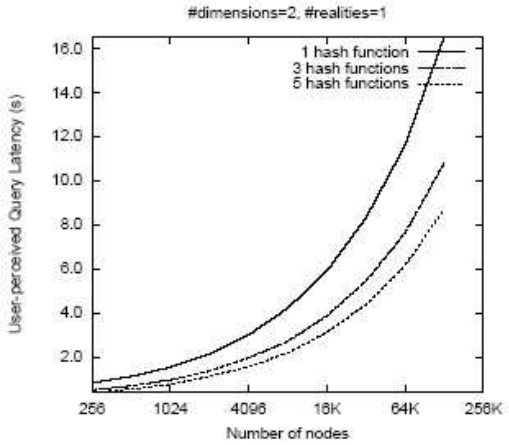

Figure 7: CAN performance results. [3]

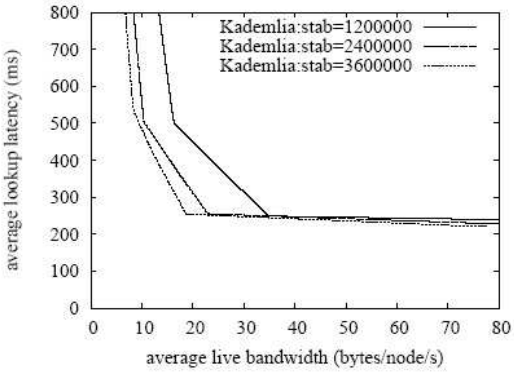

Figure 8: Kademlia lookup latencies for different stabilization times. (milliseconds) Y-axis: lookup latency in milliseconds, x-axis: traffic per node per second. [12]

4.2 Overview

This table presents the average hopcounts of different routing geometries when the network size is kept at 65,536 nodes and there are no node failures. One has to keep in mind that the results do not reflect reality where conditions are less optimal and node count is higher. [15]

Routing geometry	*Average hops*	*Median hops*
XOR	7.7	8
Ring	7.4	7
Tree	7.7	8
Butterfly	21.4	21
Hypercube	7.7	8
Hybrid	7.7	8

Table 1: *The effect of topology on lookup hopcounts for different routing geometries. [4]*

The following table presents the O-notational performance of the four DHTs, where lookup means looking up a value corresponding a key and state means maintaining the routing information for a node. The results of the Lookup column are presented as messages to other nodes. The results of the State column are presented as the number of nodes

each node has to maintain routing information on. Other DHTs achieve O(log N) performance but CAN does better depending on how many dimensions the network overlay has.

Algorithm	*Lookup*	*State*
CAN	O($dN\frac{1}{d}$)	O(d)
Chord	O(log N)	O(log N)
Kademlia	O(log N)	O(log N)
Tapestry	O(log N)	O(log N)

Table 2: *Lookup and state performances of the selected DHTs, where N is the network overlay node count and d is the number of dimensions of CAN. [13]*

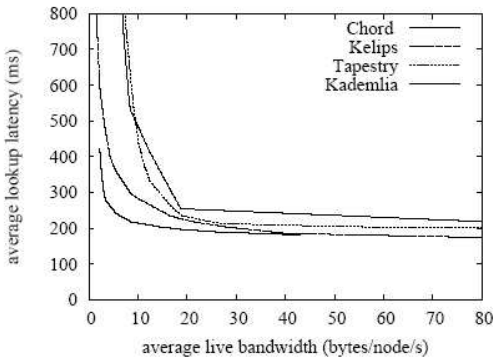

Figure 9: Average lookup latences of four DHTs in milliseconds.[4]

5 Conclusion

In this literary survey we presented four different DHT algorithms: Chord, Tapestry, CAN and Kademlia which can be implemented to provide efficient, reliable and scalable peer-to-peer overlay networks. These networks can be used to store and retrieve material and information in a decentralized manner and in such a way that there's no need for centralized network management like in the well known first generation peer-to-peer applications, which have similar content searching functions but which are mostly based on updated indexes on server and hubs. While the popularity of P2P applications increases, it's important to develop solutions that can handle massive growth of the networks and rapidly changing network topologies. These new peer protocols are making way for peer-to-peer technologies in application areas other than the well known file sharing.

6 Acknowledgements

The authors would like to thank Miika Komu and the Internetworking course staff for their help in writing the paper.

References

[1] Zhao, Huang, Stribling, Rhea, Joseph, Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *In IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS.*

[2] Rhea, Geels, Roscoe, Kubiatowicz. Handling Churn in a DHT. *University of California, Berkeley and Intel Research, Berkeley. December 2003.*

[3] Ratnasamy, Franci, Handley, Karp, Shenker. A Scalable Content-Addressable Network. *In ACM Sigcomm 2001, San Diego, CA, Aug. 2001.*

[4] Stoica, Morris, Karger, Kaashoek, Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. *In the Proceedings of ACM SIGCOMM 2001. August 2001.*

[5] Rowstron, Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *November 2001.*

[6] Balakrishnan, Kaashoek, Karger, Morris, Stoica. Looking up data in P2P systems. *In COMMUNICATIONS OF THE ACM February 2003/Vol. 46, No. 2. February 2003.*

[7] Maymounkov, Mazières. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. *In ACM Computing Surveys archive Volume 36, Issue 4. December 2004.*

[8] Sit, Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. *2002.*

[9] Ramasubramanian, Sirer. Proactive Caching for Better than Single-Hop Lookup Performance. *Cornell University, Computer Science Department Technical Report TR-2004-1931. February 2004.*

[10] Jain, Mahajan, Wetherall. A Study of the Performance Potential of DHT-based Overlays. *In Proceedings of USITS Š03: 4th USENIX Symposium on Internet Technologies and Systems. March 2003.*

[11] Castro, Costa, Rowstron. Performance and dependability of structured peer-to-peer overlays. *December 2003.*

[12] Li, Stribling, Gil, Morris, Kaashoek. Comparing the performance of distributed hash tables under churn. *February 2004.*

[13] Ramasubramanian, Sirer. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. *In 1st Symposium on Networked Systems Design and Implementation, 2004.*

[14] Dabek, Li, Sit, Robertson, Kaashoek, Morris. Designing a DHT for low latency and high throughput. *In the Proceedings of the 1st Symposium on Networked Systems Design and Implementation. March 2004.*

[15] K. Gummadi, R. Gummadi, Gribble, Ratnasamy, Shenker, Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. *In Proceedings of ACM SIGCOMM, Aug. 2003.*

[16] UC Berkeley Computer Science Division. The OceanStore Project. *http://oceanstore.cs.berkeley.edu/ Referred: 15.04.2005.*

[17] UC Berkeley Computer Science Division. The OceanStore Project at SourceForge.net *http://oceanstore.sourceforge.net/ Referred: 15.04.2005.*

[18] Tang,Xu,Mahalingam. pSearch: Information Retrieval in Structured Overlays *In ACM SIGCOMM Computer Communication Review, Volume 33, Issue 1 (Jan 2003)*