Distributed Hash Tables and Their Implementation

Denisolt Shakhbulatov
New York, USA
dshakhbu@nyit.edu

SeoHee Iris Park
New York, USA
spark26@nyit.edu

Abstract—A Distributed Hash Table(DHT) is a class of decentralized distributed systems of the search service, that works similar to hash tables. The structure of an associate key-value is stored in the DHT, while each node can look up the value associated with the given key. Responsibility for the maintaining the connection between the name and the value is distributed between the nodes, which means that the change in the set of participants is the reason for the minimum number of disruption. This technology makes it easier to scale the DHT and keep a track of added and removed nodes. Denisolt Shakhbulatov and Seohee Park will be looking at the implementation in Bittorent distributed trackers.

Index Terms—Data Structure, Peer to Peer, DHT, Distributed Computing

I. Overall about DHT

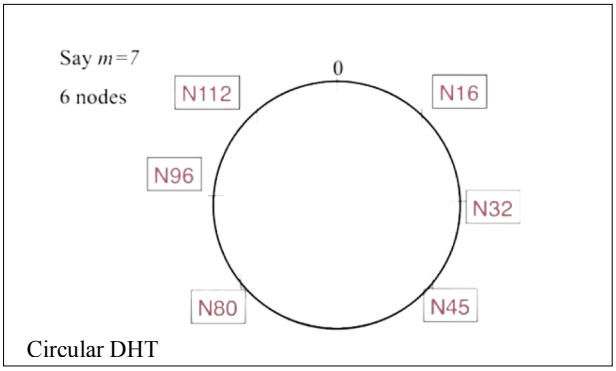
Distributed Hash Tables are very similar to the regular Hash Tables. Motivated by the problems that the Peer to Peer content distribution systems like Napster, Gnutella, Freenet and other face, researches started thinking of ways how to make P2P systems more secure, decentralized and scalable. [1]

All these goals were finally achieved with DHT by requiring a node interact with only few other nodes on the network. Which made it possible for nodes to communicate with each other faster since it would join the network, quickly update the networks state and leave.[1] Objects in DHT are files that have keys(mostly file name). Instead of storing objects into buckets like in regular Hash Table, you store them into nodes (hosts) in the cluster, which can be distributed through out the world.

The idea of DHT is to distribute the either content or location of the content on all nodes. First step is mapping. There are two ways: Direct mapping and indirect. In direct mapping each node will have an actual value stored. [1]The main issue of direct mapping is that it is not flexible for large content. Indirect mapping is more flexible however it does require an additional step to get to the stored object. In indirect mapping each node stores a tuple of key(which is hashed the same way as direct maping) and value(which is often a real adress, where object is stored).

DHT's desirable properties are: keys maped evenly to all nodes, each node maintains information about few other nodes, key can simply be found by querying and node arrival or departure will only affect few nodes.[2]

Implementation of the DHT has been done by many different research teams. Since DHT is mainly used in P2P, we will be looking into implementation on P2P Systems. Most popular implementations are: Pastry (Microsoft Research, Rice University), Chord (UC Berkeley, MIT), CAN (UC Berkeley, ICSI) and Tapestry. This paper will be based on the Chord implementation. Chord uses consistent hashing (applying SHA-1 to IP-address and port). It will return a 160bit String. After user truncates the string to m bits, where m is the system parameter, which will return a peer id (number between 0 and 2^m-1), peer id



Circular DHT

mod 2^m. If M is large enough, the number of possible conflicts will decrease. Each of the nodes in P2P knows the IP-address and port number of its successor (clockwise neighbour) and predecessor. Second peer pointer is Finger Tables. Filenames are also mapped using SHA-1 and gets truncated. Depending on the result of the operation, the file will be stored in the first peer's storage that is immediately clockwise to that point. same as accessing a memory address using a hash table, but in a network if you will. [4]

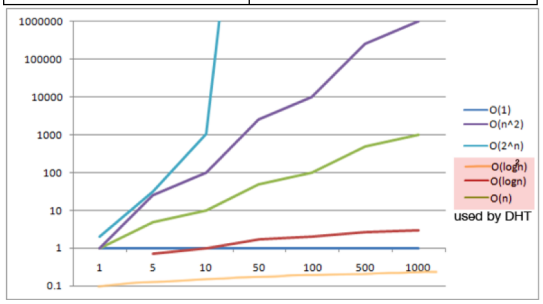
II. Supported Operations of DHT

DHT supports operations such as: finger[k] - identifier, find_successor – returns the next node from the node in question on the identifier ring, closest_predecessor. There are 2 types of lookup operation: basic lookup, when algorithm hops node by node via successor until finds destination id - O(n), also there is efficient lookup, which routes via distant "finger" nodes - O(log n). Join operation is same as insert operation - $O(\log^2 N)$. Leave is the delete operation for DHT - $O(\log^2 N)$. Other supported operations are: stabilize, notify, fix_fingers, check_predecessor. [2]

III. Asymptotic Growth Functions of Major Operations

Find Operation can take either O(n) or O(log N) depending on which type of lookup is used. Insertion is $O(\log^2 N)$, same as deletion method. [2]

Insert	$O(\log^2 N)$
Delete	$O(\log^2 N)$
Find	O(log N) or O(N)



IV. Application of DHT

Most popular application of DHT is using it in Distributed storages, such as BitTorrent. Peer IP addresses are stored using infohash as the key. Since BitTorent allows to look up and store IP addresses in the DHT through infohash, 'get' request will look up an infohash in the torrent and return the IP addresses, while 'put' request will store an IP address. Peers are randomly assigned to store values belonging to little parts of the of the key space, it will ensure that keys are evenly distributed across participating peers. [3]

References

[1] CENL, "Distributed Hash Tables – DHT" pp. 2-77, January 2006¬

[2] V.Richard, "Lecture on Distributed Hash Tables" pp. 10–13, December 2015

[3] M.Leiva-Gomez, "MTE Explains: How BitTorrent DHT Peer Discovery Works" January 2013

[4] Ion Stoica, Roberr Morris, David Kargar, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *In Proceedings of the ACM SIGCOMM '01 Conference*, California, August 2001.