Simple basic explanation of a Distributed Hash Table (DHT)

Asked 16 years, 2 months ago Modified 3 months ago

Viewed 83k times

197

Could any one give an explanation on how a DHT works?

Nothing too heavy, just the basics.

theory p2p dht

Share

Improve this question

Follow

edited Nov 10, 2015 at 10:46

nbro

16k ● 34 ● 119 ● 212

asked Sep 27, 2008 at 20:08

Gustavo Carreno

9,749 ● 13 ● 48 ● 78

4 Answers

Sorted by: Highest score (default)

Ok, they're fundamentally a pretty simple idea. A DHT gives you a dictionary-like interface, but the nodes are distributed across the network. The trick with DHTs is that the node that gets to store a particular key is found by hashing that key, so in effect your hash-table buckets are now independent nodes in a network.

This gives a lot of fault-tolerance and reliability, and possibly some performance benefit, but it also throws up a lot of headaches. For example, what happens when a node leaves the network, by failing or otherwise? And how do you redistribute keys when a node joins so that the load is roughly balanced. Come to think of it, how do you evenly distribute keys anyhow? And when a node joins, how do you avoid rehashing everything? (Remember you'd have to do this in a normal hash table if you increase the number of buckets).

One example DHT that tackles some of these problems is a logical ring of n nodes, each taking responsibility for 1/n of the keyspace. Once you add a node to the network, it finds a place on the ring to sit between two other nodes, and takes responsibility for some of the keys in its sibling nodes. The beauty of this approach is that none of the other nodes in the ring are affected; only the two sibling nodes have to redistribute keys.

For example, say in a three node ring the first node has keys 0-10, the second 11-20 and the third 21-30. If a fourth node comes along and inserts itself between nodes 3 and 0 (remember, they're in a ring), it can take

responsibility for say half of 3's keyspace, so now it deals with 26-30 and node 3 deals with 21-25.

There are many other overlay structures such as this that use content-based routing to find the right node on which to store a key. Locating a key in a ring requires searching round the ring one node at a time (unless you keep a local look-up table, problematic in a DHT of thousands of nodes), which is O(n)-hop routing. Other structures - including augmented rings - guarantee O(log n)-hop routing, and some claim to O(1)-hop routing at the cost of more maintenance.

Read the wikipedia page, and if you really want to know in a bit of depth, check out this coursepage at Harvard which has a pretty comprehensive reading list.

Share Improve this answer

Follow

answered Sep 27, 2008 at 20:59

HenryR

8,509 ● 7 ● 36 ● 39

27 +1 Good answer. What you mean in third paragraph ("One example DHT that tackles some of these problems is a logical ring of n nodes") is Consistent Hashing. It's a really interesting topic, used in Apache Cassandra, a Distributed database created by Facebook. Link to paper (worth reading it): cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf – santiagobasulto Sep 2, 2011 at 13:00

5 A ring-based lookup protocol that is pretty easy to understand is Chord: pdos.csail.mit.edu/papers/chord:sigcomm01 – ThomasWeiss Jan 8, 2014 at 3:55

Can you please elaborate on how key-value are stored on a node? Will it be some form of hash table or a DB?.
– Wand Maker Jul 18, 2016 at 20:02

@HenryR, Isn't "node ring" simply a tree structure?
– Pacerier Nov 3, 2017 at 22:42

1 The Uni of Illinois teaches the chord protocol every semester as part of their distributed systems class if anyone wants more reading material - courses.engr.illinois.edu/ece428/sp2018/lectures.html
– Siddhartha Apr 12, 2018 at 6:42

11

DHTs provide the same type of interface to the user as a normal hashtable (look up a value by key), but the data is distributed over an arbitrary number of connected nodes. Wikipedia has a good basic introduction that I would essentially be regurgitating if I write more -

http://en.wikipedia.org/wiki/Distributed_hash_table

Share Improve this answer

Follow

edited Mar 1, 2015 at 15:22

Igor Popov
10.1k ● 7 ● 56 ● 70

answered Sep 27, 2008 at 20:12

Peter
29.8k ● 22 ● 91 ● 126

10

I'd like to add onto HenryR's useful answer as I just had an insight into consistent hashing. A normal/naive hash lookup is a function of two variables, one of which is the

number of buckets. The beauty of consistent hashing is that we eliminate the number of buckets "n", from the equation.

In naive hashing, first variable is the key of the object to be stored in the table. We'll call the key "x". The second variable is is the number of buckets, "n". So, to determine which bucket/machine the object is stored in, you have to calculate: hash(x) mod(n). Therefore, when you change the number of buckets, you also change the address at which almost every object is stored.

Compare this to consistent hashing. Let's define "R" as the range of a hash function. R is just some constant. In consistent hashing, the address of an object is located at hash(x)/R. Since our lookup is no longer a function of the number of buckets, we end up with less remapping when we change the number of buckets.

And as Pacerier points out in the comments, you then take the modulus of the number of buckets.

Bucket address = hash(x)/(R * n) where n is the number of buckets.

http://michaelnielsen.org/blog/consistent-hashing/

edited Sep 20 at 4:40

answered Apr 28, 2016 at 21:45

thebiggestlebowski
2,769 ● 1 ● 34 ● 32

2 You'd still need to mod anyway isn't it? Say you got 3 servers. `hash(x)/R` gives you 34500. **You still need to do 34500 % 3**. – Pacerier Nov 3, 2017 at 22:48

Your blogpost is unclear btw, you should list of the step by step snapshot of a **working example** where nodes are added and removed along with rows that are added and removed. – Pacerier Nov 3, 2017 at 23:10

4

The core of a DHT is a hash table. Key-value pairs are stored in DHT and a value can be looked up with a key. The keys are unique identifiers to values that can range from blocks in a blockchain to addresses and to documents.

What differentiates a DHT from a normal hash table is the fact that storage and lookup on DHT are distributed across multiple (can be millions) nodes or machines. This very characteristic of DHT makes it look like distributed databases used for storage and retrieval. There is no master-slave hierarchy or a centralized control among the participating nodes. All the nodes are treated as peers.

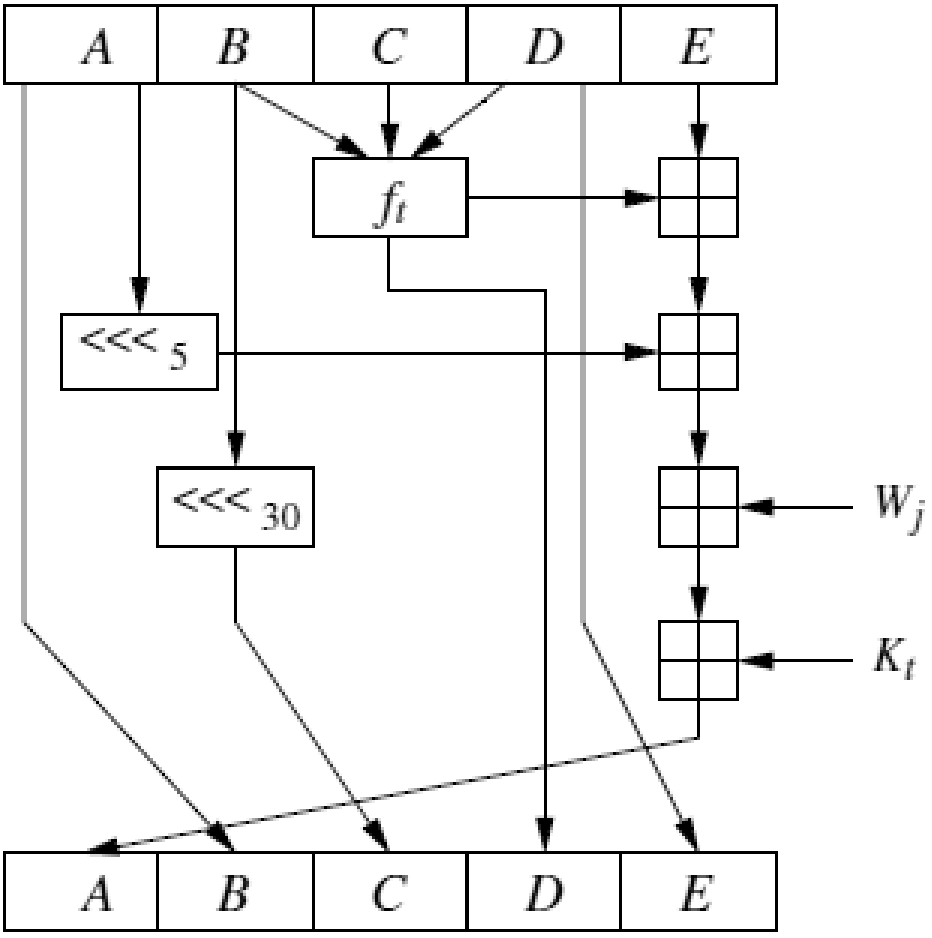
DHT provides freedom to the participating nodes such that the nodes can join or leave the network anytime. Due to this reason, DHTs are widely used in Peer-to-Peer (P2P) networks. In fact, part of the motivation behind the research of DHT stems from its usage in P2P networks.

Characteristics of DHT

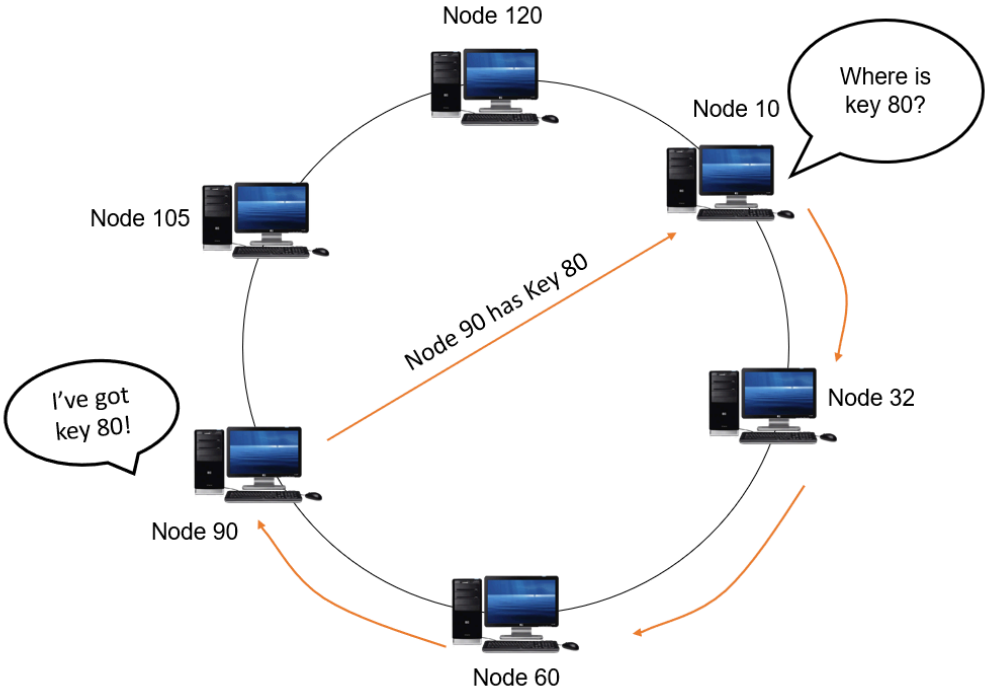
1. Decentralized: Since there is no central authority or coordination
2. Scalable: The system can easily scale up to millions of nodes
3. Fault-tolerant: DHT replicates the data storage on all the nodes. Therefore, even if one node leaves the network, it should not affect other nodes in the network.

Let’s see how lookup happens in a popular DHT protocol like Chord. Consider a circular doubly-linked list of nodes. Each node has a reference pointer to the node previous as well as next to it. The node next to the node in question is called the successor. The node that is previous to the node in question is called the predecessor.

Speaking in terms of a DHT, each node has a unique node ID of k bits and these nodes are arranged in the increasing order of their node IDs. Assume these nodes are arranged in a ring structure called identifier ring. For each node, the successor has the shortest distance clockwise away. For most nodes, this is the node whose ID is closest to but still greater than the current node’s ID. To find out the node appropriate for a particular key, first hash the key K and all the nodes to exactly k bits using consistent hashing techniques like SHA-1.



Start at any point in the ring and traverse clockwise till you catch the node whose node ID is closer to the key K, but can be greater than K. This node is the one responsible for storage and lookup for that particular key.



In an iterative style of lookup, each node Q queries its successor node for KV (key-value) pair. If the queried node does not have the target key, it will return a set of nodes S that can be closer to the target. The querying node Q then queries the nodes in S which are closer to itself. This continues until either the target KV pair is returned or when there are no more nodes to query.

This lookup is very suitable for an ideal scenario where all the nodes have a perfect uptime. But how to handle scenarios when nodes leave the network either intentionally or by failure? This calls for the need for a robust join/leave protocol.

Popular DHT protocols and implementations

1. Chord
2. Kademlia
3. Apache Cassandra
4. Koorde TomP2P
5. Voldemort

References:

1. https://en.wikipedia.org/wiki/Distributed_hash_table
2. https://steffikj19.medium.com/dht-demystified-77dd31727ea7
3. https://www.linuxjournal.com/article/6797

 edited Mar 27, 2021 at 17:17

Follow

answered Mar 27, 2021 at 17:11

Steffi Keran Rani J
4,083 4 39 60