Distributed hash table

From Wikipedia, the free encyclopedia

Distributed hash tables (**DHTs**) are a class of decentralized distributed systems that provide a lookup service similar to a hash table: (*key*, *value*) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

DHTs form an infrastructure that can be used to build more complex services, such as distributed file systems, peer-to-peer file sharing and content distribution systems, cooperative web caching, multicast, anycast, domain name services, and instant messaging. Notable distributed networks that use DHTs include BitTorrent's distributed tracker, the eDonkey network, the Storm botnet, YaCy, and the Coral Content Distribution Network.

Contents

History

DHT research was originally motivated, in part, by peer-to-peer systems such as Napster, Gnutella, and Freenet, which took advantage of resources distributed across the Internet to provide a single useful application. In particular, they took advantage of increased bandwidth and hard disk capacity to provide a file sharing service.

These systems differed in how they *found* the data their peers contained:

- Napster had a central index server: each node, upon joining, would send a list of locally held files to the server, which would perform searches and refer the querier to the nodes that held the results. This central component left the system vulnerable to attacks and lawsuits.
- Gnutella and similar networks moved to a flooding query model—in essence, each search would result in a message being broadcast to every other machine in the network. While avoiding a single point of failure, this method was significantly less efficient than Napster.
- Finally, Freenet was also fully distributed, but employed a heuristic key based routing in which each file was associated with a key, and files with similar keys tended to cluster on a similar set of nodes. Queries were likely to be routed through the network to such a cluster

without needing to visit many peers. However, Freenet did not guarantee that data would be found.

Distributed hash tables use a more structured key based routing in order to attain both the decentralization of Gnutella and Freenet, and the efficiency and guaranteed results of Napster. One drawback is that, like Freenet, DHTs only directly support exact-match search, rather than keyword search, although that functionality can be layered on top of a DHT.

The first four DHTs—CAN, Chord,[1] Pastry, and Tapestry—were introduced about the same time in 2001. Since then this area of research has been quite active. Outside academia, DHT technology has been adopted as a component of BitTorrent and in the Coral Content Distribution Network.

Properties

DHTs characteristically emphasize the following properties:

- Decentralization: the nodes collectively form the system without any central coordination.
- Scalability: the system should function efficiently even with thousands or millions of nodes.
- Fault tolerance: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.

A key technique used to achieve these goals is that any one node needs to coordinate with only a few other nodes in the system – most commonly, $\Theta(\log n)$ of the n participants (see below) – so that only a limited amount of work needs to be done for each change in membership.

Some DHT designs seek to be secure against malicious participants and to allow participants to remain anonymous, though this is less common than in many other peer-to-peer (especially file sharing) systems; see anonymous P2P.

Finally, DHTs must deal with more traditional distributed systems issues such as load balancing, data integrity, and performance (in particular, ensuring that operations such as routing and data storage or retrieval complete quickly).

Structure

The structure of a DHT can be decomposed into several main components.[2][3] The foundation is an abstract **keyspace**, such as the set of 160-bit strings. A **keyspace partitioning** scheme splits ownership of this keyspace among the participating nodes. An **overlay network** then connects the nodes, allowing them to find the owner of any given key in the keyspace.

Once these components are in place, a typical use of the DHT for storage and retrieval might proceed as follows. Suppose the keyspace is the set of 160-bit strings. To store a file with given *filename* and *data* in the DHT, the SHA1 hash of *filename* is found, producing a 160-bit key *k*, and a message *put*(*k*,*data*) is sent to any node participating in the DHT. The message is forwarded from node to node through the overlay network until it reaches the single node responsible for key *k* as specified by the keyspace partitioning, where the pair (*k*,*data*) is stored. Any other client can then retrieve the contents of the file by again hashing *filename* to produce *k* and asking any DHT node to find the data associated with *k* with a message *get*(*k*). The message will again be routed through the overlay to the node responsible for *k*, which will reply with the stored *data*.

The keyspace partitioning and overlay network components are described below with the goal of

capturing the principal ideas common to most DHTs; many designs differ in the details.

Keyspace partitioning

Most DHTs use some variant of consistent hashing to map keys to nodes. This technique employs a function $\delta(k_1,k_2)$ which defines an abstract notion of the *distance* from key k_1 to key k_2, which is unrelated to geographical distance or network latency. Each node is assigned a single key called its *identifier* (ID). A node with ID *i* owns all the keys for which *i* is the closest ID, measured according to δ.

> **Example.** The Chord DHT treats keys as points on a circle, and $\delta(k_1,k_2)$ is the distance traveling clockwise around the circle from k_1 to k_2. Thus, the circular keyspace is split into contiguous segments whose endpoints are the node identifiers. If i_1 and i_2 are two adjacent IDs, then the node with ID i_2 owns all the keys that fall between i_1 and i_2.

Consistent hashing has the essential property that removal or addition of one node changes only the set of keys owned by the nodes with adjacent IDs, and leaves all other nodes unaffected. Contrast this with a traditional hash table in which addition or removal of one bucket causes nearly the entire keyspace to be remapped. Since any change in ownership typically corresponds to bandwidth-intensive movement of objects stored in the DHT from one node to another, minimizing such reorganization is required to efficiently support high rates of churn (node arrival and failure).

Overlay network

Each node maintains a set of links to other nodes (its *neighbors* or routing table). Together these links form the overlay network. A node picks its neighbors according to a certain structure, called the network's topology.

All DHT topologies share some variant of the most essential property: for any key *k*, the node either owns *k* or has a link to a node that is *closer* to *k* in terms of the keyspace distance defined above. It is then easy to route a message to the owner of any key *k* using the following greedy algorithm: at each step, forward the message to the neighbor whose ID is closest to *k*. When there is no such neighbor, then we must have arrived at the closest node, which is the owner of *k* as defined above. This style of routing is sometimes called key based routing.

Beyond basic routing correctness, two important constraints on the topology are to guarantee that the maximum number of hops in any route (route length) is low, so that requests complete quickly; and that the maximum number of neighbors of any node (maximum node degree) is low, so that maintenance overhead is not excessive. Of course, having shorter routes requires higher maximum degree. Some common choices for maximum degree and route length are as follows, where *n* is the number of nodes in the DHT, using Big O notation:

- Degree $O(1)$, route length $O(\log n)$
- Degree $O(\log n)$, route length $O(\log n / \log\log n)$
- Degree $O(\log n)$, route length $O(\log n)$
- Degree $O(n^{1/2})$, route length $O(1)$

The third choice is the most common, even though it is not quite optimal in terms of degree/route length tradeoff, because such topologies typically allow more flexibility in choice of neighbors. Many DHTs use that flexibility to pick neighbors which are close in terms of latency in the physical underlying network.

Maximum route length is closely related to diameter: the maximum number of hops in any shortest path between nodes. Clearly the network's route length is at least as large as its diameter, so DHTs are limited by the degree/diameter tradeoff[4] which is fundamental in graph theory. Route length can be greater than diameter since the greedy routing algorithm may not find shortest paths.[5]

Algorithms for overlay networks

Aside from routing, there exist many algorithms which exploit the structure of the overlay network for sending a message to all nodes, or a subset of nodes, in a DHT.[6] These algorithms are used by applications to do overlay multicast, range queries, or to collect statistics.

Real world DHTs and their differences and improvements over basic schemes

Most notable differences encountered in "real world" DHT implementations include at least the following:

- The address space is a parameter of DHT. Several real world DHTs use 128 bit or 160 bit key space
- Some real-world DHTs use hash functions other than SHA1.
- In the real world the key *k* could be a hash of a file's *content* rather than a hash of a file's *name*, so that renaming of the file does not prevent users from finding it.
- Some DHTs may also publish objects of different types. For example, key *k* could be node *ID* and associated data could describe how to contact this node. This allows publication of presence information and often used in IM applications, etc. In simplest case *ID* is just a random number which is directly used as key *k* (so in 160-bits DHT *ID* will be a 160 bits number, usually randomly chosen). In some DHTs publishing of nodes IDs is also used to optimize DHT operations.
- Key *k* could be stored to more than exactly one node corresponding to such key to cause redundancy and improve DHT reliability. Usually rather than selecting one node, real world DHT algorithm selects *i* suitable nodes and it is an implementation specific parameter of DHT reflecting DHT's redundancy. In such DHT designs nodes agree to handle certain keyspace range which is even sometimes not a hardcoded value but dynamically chosen
- Some advanced DHTs like Kademlia are rather doing iterative lookups through DHT first to select set of suitable nodes and only sending *put*(*k*,*data*) message to such nodes therefore drastically reducing useless traffic since publish messages are only sent to nodes which are apparently suitable for storing such key *k* and iterative lookups are only touching few nodes rather than whole DHT while useless forwarding is eliminated. In such DHTs forwarding of *put*(*k*,*data*) message may only occur as part of self-healing algorithm: if target node receives *put*(*k*,*data*) message but believes that key *k* is out of handled range and closer (in terms of DHT keyspace) node is known, message is forwarded to such node. Otherwise data are indexed locally. This leads to somewhat self-balancing DHT behavior. Of course such algorithm requires nodes to publish their presence data in DHT to allow mentioned iterative lookups of nodes itself via DHT before sending *put*(*k*,*data*) messages.

Examples

DHT protocols and implementations

- CAN (Content Addressable Network)
- Chord

- Kademlia
- Pastry
- P-Grid
- Tapestry

Applications employing DHTs

- BitTorrent: File distribution. BitTorrent optionally uses a DHT as a distributed tracker to provide rendezvous between clients downloading a particular file (see BitTorrent client)
- The Circle: File sharing and chat
- Codeen: Web caching
- Coral Content Distribution Network
- Dijjer: Freenet-like distribution network
- eMule: File sharing
- FAROO: Peer-to-peer web search engine
- GNUnet: Freenet-like distribution network including a DHT implementation [1]
- I2P: Anonymous network
- JXTA: Opensource P2P platform
- LimeWire: File sharing; includes the Mojito DHT
- NEOnet: File sharing
- Overnet: File sharing
- Warez P2P: File sharing
- YaCy: distributed search engine

See also

- memcached: a high-performance, distributed memory object caching system
- Prefix Hash Tree: Sophisticated querying over DHTs

References

1. ^ Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in P2P systems. In Communications of the ACM, February 2003.
2. ^ Moni Naor and Udi Wieder. Novel Architectures for P2P Applications: the Continuous-Discrete Approach. Proc. SPAA, 2003.
3. ^ Gurmeet Singh Manku. Dipsea: A Modular Distributed Hash Table. Ph. D. Thesis (Stanford University), August 2004.
4. ^ The (Degree,Diameter) Problem for Graphs
5. ^ Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy Neighbor's Neighbor: the Power of Lookahead in Randomized P2P Networks. Proc. STOC, 2004.
6. ^ Ali Ghodsi. Distributed k-ary System: Algorithms for Distributed Hash Tables. KTH-Royal Institute of Technology, 2006.

External links

- Distributed Hash Tables, Part 1 by Brandon Wiley.
- Distributed Hash Tables links Carles Pairot's Page on DHT and P2P research
- GemStone's GemFire supports a DHT with optional redundancy for high availability
- Tangosol Coherence includes a structure similar to a DHT, though all nodes have knowledge of the other participants
- kademlia.scs.cs.nyu.edu Archive.org snapshots of kademlia.scs.cs.nyu.edu
- Open DHT: A publicly accessible distributed hash table (DHT) service.
- Cacheonix uses an approach similar to DHT with added replication for better failure-tolerance
- Hazelcast: P2P and partitioned DHT implementation.

Retrieved from "http://en.wikipedia.org/wiki/Distributed_hash_table"
Categories: Distributed computing | Distributed data sharing | File sharing