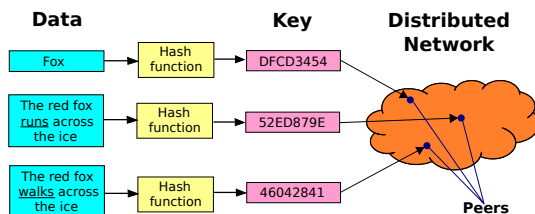


Distributed hash table

A **distributed hash table (DHT)** is a class of a decentralized **distributed system** that provides a lookup service similar to a **hash table**: (*key*, *value*) pairs are stored in a 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 a DHT 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 **anycast**, cooperative Web caching, distributed file systems, domain name services, instant messaging, multicast, and also **peer-to-peer** file sharing and content distribution systems. Notable distributed networks that use DHTs include BitTorrent's distributed tracker, the Coral Content Distribution Network, the Kad network, the Storm botnet, the Tox instant messenger, Freenet and the YaCy search engine.



Distributed hash tables

1 History

DHT research was originally motivated, in part, by **peer-to-peer** systems such as **Freenet**, **gnutella**, **BitTorrent** and **Napster**, 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 located the data offered by their peers. Napster, the first large-scale P2P content delivery system, required 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 queries 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. Later versions of Gnutella clients moved to a dynamic querying model which vastly improved efficiency.

Freenet is fully distributed, but employs a heuristic **key-based routing** in which each file is associated with a key, and files with similar keys tend to cluster on a similar set of nodes. Queries are likely to be routed through the network to such a cluster without needing to visit many peers.^[1] However, Freenet does not guarantee that data will be found.

Distributed hash tables use a more structured key-based routing in order to attain both the decentralization of Freenet and gnutella, 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 Freenet's routing algorithm can be generalized to any key type where a closeness operation can be defined.^[2]

In 2001, four systems—CAN,^[3] Chord,^[4] Pastry, and Tapestry—ignited DHTs as a popular research topic. A project called the Infrastructure for Resilient Internet Systems (Iris) was funded by a \$12 million grant from the US National Science Foundation in 2002.^[5] Researchers included Sylvia Ratnasamy, Ion Stoica, Hari Balakrishnan and Scott Shenker.^[6] Outside academia, DHT technology has been adopted as a component of BitTorrent and in the Coral Content Distribution Network.

2 Properties

DHTs characteristically emphasize the following properties:

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

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, $O(\log n)$ 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^[7] 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).

3 Structure

The structure of a DHT can be decomposed into several main components.^{[8][9]} 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 index a file with given filename and data in the DHT, the **SHA-1** hash of filename is generated, 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. That node then stores the key and the data. 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.

3.1 Keyspace partitioning

Most DHTs use some variant of **consistent hashing** or **rendezvous hashing** to map keys to nodes. The two algorithms appear to have been devised independently and simultaneously to solve the distributed hash table problem.

Both consistent hashing and rendezvous hashing have 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).

3.1.1 Consistent hashing

Consistent hashing employs a function $\delta(k_1, k_2)$ that defines an abstract notion of the distance between the keys k_1 and 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_x owns all the keys k_m for which i_x is the closest ID, measured according to $\delta(k_m, i_x)$.

For example, the **Chord DHT** uses consistent hashing, which 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, with a shorter clockwise distance from i_1 to i_2 , then the node with ID i_2 owns all the keys that fall between i_1 and i_2 .

3.1.2 Rendezvous hashing

In **rendezvous hashing**, also called highest random weight hashing, all clients use the same hash function $h()$ (chosen ahead of time) to associate a key to one of the n available servers. Each client has the same list of identifiers $\{S_1, S_2, \dots, S_n\}$, one for each server. Given some key k , a client computes n hash weights $w_1 = h(S_1, k)$, $w_2 = h(S_2, k)$, ..., $w_n = h(S_n, k)$. The client associates that key with the server corresponding to the highest hash weight for that key. A server with ID S_x owns all the keys k_m for which the hash weight $h(S_x, k_m)$ is higher than the hash weight of any other node for that key.

3.1.3 Locality-preserving hashing

Locality-preserving hashing ensures that similar keys are assigned to similar objects. This can enable a more efficient execution of range queries. Self-Chord^[10] decouples object keys from peer IDs and sorts keys along the ring with a statistical approach based on the **swarm intelligence** paradigm. Sorting ensures that similar keys are stored by neighbour nodes and that discovery procedures, including range queries, can be performed in logarithmic time.

3.2 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 , each node either has a node ID that owns k or has a link to a node whose node ID 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** (that is not necessarily globally optimal): 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:

The most common choice, $O(\log n)$ degree/route length, is not optimal in terms of degree/route length tradeoff, but such topologies typically allow more flexibility in choice of neighbors. Many DHTs use that flexibility to pick neighbors that 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 worst case route length is at least as large as its diameter, so DHTs are limited by the degree/diameter tradeoff^[11] that is fundamental in **graph theory**. Route length can be greater than diameter, since the greedy routing algorithm may not find shortest paths.^[12]

3.3 Algorithms for overlay networks

Aside from routing, there exist many algorithms that exploit the structure of the overlay network for sending a message to all nodes, or a subset of nodes, in a DHT.^[13] These algorithms are used by applications to do **overlay multicast**, range queries, or to collect statistics. Two systems that are based on this approach are Structella,^[14] which implements flooding and random walks on a Pastry overlay, and DQ-DHT,^[15] which implements a dynamic querying search algorithm over a Chord network.

4 Security

Because of the decentralization, fault tolerance, and scalability of DHTs, they are inherently more resilient against a hostile attacker than a typical centralized system.

Open systems for **distributed data storage** that are robust against massive hostile attackers are feasible.^[16]

A DHT system that is carefully designed to have **Byzantine fault tolerance** can defend against a **Sybil attack**.^{[17][18]}

5 DHT implementations

Most notable differences encountered in practical instances of 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 SHA-1.
- 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* to provide **content-addressable storage**, 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 the 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 the simplest case, *ID* is just a random number that is directly used as key k (so in a 160-bit DHT *ID* will be a 160-bit number, usually randomly chosen). In some DHTs, publishing of nodes' IDs is also used to optimize DHT operations.
- Redundancy can be added to improve reliability. The $(k, data)$ key pair can be stored in more than one node corresponding to the key. Usually, rather than selecting just one node, real world DHT algorithms select i suitable nodes, with i being an implementation-specific parameter of the DHT. In some DHT designs, nodes agree to handle a certain keyspace range, the size of which may be chosen dynamically, rather than hard-coded.
- Some advanced DHTs like **Kademlia** perform iterative lookups through the DHT first in order to select a set of suitable nodes and send $put(k, data)$ messages only to those nodes, thus drastically reducing useless traffic, since published messages are only sent to nodes that seem suitable for storing the key k ; and iterative lookups cover just a small set of nodes rather than the entire DHT, reducing useless forwarding. In such DHTs, forwarding of $put(k, data)$

messages may only occur as part of a self-healing algorithm: if a target node receives a $put(k, data)$ message, but believes that k is out of its handled range and a closer node (in terms of DHT keyspace) is known, the message is forwarded to that node. Otherwise, data are indexed locally. This leads to a somewhat self-balancing DHT behavior. Of course, such an algorithm requires nodes to publish their presence data in the DHT so the iterative lookups can be performed.

6 Examples


7 See also

- **Couchbase Server**: a persistent, replicated, clustered distributed object storage system compatible with memcached protocol.
- **Memcached**: a high-performance, distributed memory object caching system.
- **Prefix hash tree**: sophisticated querying over DHTs.
- **Merkle tree**: tree having every non-leaf node labelled with the hash of the labels of its children nodes.
- Most distributed data stores employ some form of DHT for lookup.

8 References

- [1] *Searching in a Small World Chapters 1 & 2* (PDF), retrieved 2012-01-10
- [2] "Section 5.2.2", *A Distributed Decentralized Information Storage and Retrieval System* (PDF), retrieved 2012-01-10
- [3] Ratnasamy; et al. (2001). "A Scalable Content-Addressable Network" (PDF). In Proceedings of ACM SIGCOMM 2001. Retrieved 2013-05-20.
- [4] 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.
- [5] David Cohen (October 1, 2002). "New P2P network funded by US government". *New Scientist*. Retrieved November 10, 2013.
- [6] "MIT, Berkeley, ICSI, NYU, and Rice Launch the IRIS Project". *Press release*. MIT. September 25, 2002. Retrieved November 10, 2013.
- [7] Guido Urdaneta, Guillaume Pierre and Maarten van Steen. A Survey of DHT Security Techniques. ACM Computing Surveys 43(2), January 2011.
- [8] Moni Naor and Udi Wieder. Novel Architectures for P2P Applications: the Continuous-Discrete Approach. Proc. SPAA, 2003.
- [9] Gurmeet Singh Manku. Dipsea: A Modular Distributed Hash Table. Ph. D. Thesis (Stanford University), August 2004.
- [10] Agostino Forestiero, Emilio Leonardi, Carlo Mastroianni and Michela Meo. Self-Chord: a Bio-Inspired P2P Framework for Self-Organizing Distributed Systems. IEEE/ACM Transactions on Networking, 2010.
- [11] *The (Degree,Diameter) Problem for Graphs*, Maite71.upc.es, retrieved 2012-01-10
- [12] Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy Neighbor's Neighbor: the Power of Lookahead in Randomized P2P Networks. Proc. STOC, 2004.
- [13] Ali Ghodsi. Distributed k-ary System: Algorithms for Distributed Hash Tables Archived May 22, 2007, at the Wayback Machine.. KTH-Royal Institute of Technology, 2006.
- [14] Miguel Castro, Manuel Costa, and Antony Rowstron. Should we build Gnutella on a structured overlay?. Computer Communication Review, 2004.
- [15] Domenico Talia and Paolo Trunfio. Enabling Dynamic Querying over Distributed Hash Tables. Journal of Parallel and Distributed Computing, 2010.
- [16] Baruch Awerbuch, Christian Scheideler. "Towards a scalable and robust DHT". 2006. doi:10.1145/1148109.1148163
- [17] Maxwell Young; Aniket Kate; Ian Goldberg; Martin Karsten. "Practical Robust Communication in DHTs Tolerating a Byzantine Adversary".
- [18] Natalya Fedotova; Giordano Orzetti; Luca Veltri; Alessandro Zaccagnini. "Byzantine agreement for reputation management in DHT-based peer-to-peer networks". doi:10.1109/ICTEL.2008.4652638
- [19] Tribler wiki Archived December 4, 2010, at the Wayback Machine. retrieved January 2010.
- [20] Retroshare FAQ retrieved December 2011

9 External links

- **Distributed Hash Tables, Part 1** by Brandon Wiley.
- **Distributed Hash Tables** links Carles Pairot's Page on DHT and P2P research
- **kademlia.scs.cs.nyu.edu** Archive.org snapshots of kademlia.scs.cs.nyu.edu
- Eng-Keong Lua; Crowcroft, Jon; Pias, Marcelo; Sharma, Ravi; Lim, Steve. "IEEE Survey on overlay network schemes". *CiteSeerX* 10.1.1.111.4197 : covering unstructured and structured decentralized overlay networks including DHTs (Chord, Pastry, Tapestry and others).

- Mainline DHT Measurement at Department of Computer Science, University of Helsinki, Finland.

10 Text and image sources, contributors, and licenses

10.1 Text

- **Distributed hash table** *Source:* https://en.wikipedia.org/wiki/Distributed_hash_table?oldid=770987076 *Contributors:* Bryan Derksen, The Anome, Edward, Nealmcb, TakuyaMurata, Alfio, Fcrick, Haakon, Ronz, Athymik~enwiki, Ehn, Hashar, Charles Matthews, Itai, Johnleach, GPHemsley, Jamesday, Phil Boswell, Bernhard Bauer, Yramesh, Hadal, UtherSRG, Seano1, Anthony, Diberri, Thv, SamB, DavidCary, Elf, ShaunMacPherson, Khalid hassani, ReinoutS, Naff89, Corti, Mike Rosoft, Jda, TedPavlic, Luqui, Wk muriithi, Irrbloss, Unquietwiki, Godrickwok, Mdd, Gary, Mgrinich, CyberSkull, Minority Report, Apoc2400, EmilSit, Cburnett, Karnesky, The Belgain, Tabletop, Knuckles, Meneth, Eras-mus, Sega381, Xiong Chiamiov, Enzo Aquarius, Search4Lancer, Xosé, X1987x, Intgr, Garas, Roboto de Ajvol, YurikBot, MMuzammils, Nad, Nethgurb, Cedar101, Cojoco, Eric.weigle, NeilN, Br~enwiki, Crystallina, KnightRider~enwiki, Erik Sandberg, SmackBot, Y10k, F, Pkg, Mcl, Ohnoitsjamie, Bluebot, Morte, OrangeDog, Baa, Frap, Iammisc, JonHarder, FlyHigh, G-Bot~enwiki, Harryboyles, Oskilian, Nhorton, Roger pack, Michael miceli, Anescient, M.B~enwiki, Hu12, Gpierre, Dto, DJPhazer, Imeshev, Cydebot, Neustradamus, Thijs!bot, Ssspera, Marek69, TheJosh, CosineKitty, Nazlfrag, Bubba hoteb, DerHexer, STBot, Shentino, Monkeyjunky, LordAnubisBOT, Hbpencil, OPless, Jnlin, VolkovBot, Jensocan, TelecomNut, Rogerdpack, Andy Dingley, Tomaxer, Bpringlemeir, Kbrose, Cacheonix, Alexbot, Thingg, DumZiBoT, Dsimic, MikeSchmid111, Addbot, Mabdul, Goofi1781, LaaknorBot, Ginosbot, Numbo3~bot, Jonhoo, Yobot, Ptbotgourou, Old Death, Chy168, AnomieBOT, Götz, Squalho, Xqbot, Happyrabbit, Tomdo08, Miym, Chrismiceli, RibotBOT, Xubupt, W Nowicki, Sanpitch, Sae1962, Clsin, Louperibot, RedBot, Irvine.david, Tombeo, Liamzebedee, Maix, Noodles-sb, Ingenth, Brkt, Greywiz, EmausBot, Orphan Wiki, Md4567, Allquixotic, Scgtrp, Retsejesiw, Wayne Slam, Chuispaston-Bot, JaredThornbridge, CocuBot, PetrIvanov, Mpias, Elauminri, John13535, Igstan, Zorun, AstReseach, Raghavendra.talur, Justinesherry, Khiladi 2010, ChrisGualtieri, Pinto, Wario-Man, Altered Walter, TheGuyOfDoom, Monkbot, JFreegman, Graviton-Spark, Kevin at aerospike, InternetArchiveBot, GreenC bot, Ushkin N and Anonymous: 249

10.2 Images

- **File:DHT_en.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/98/DHT_en.svg *License:* Public domain *Contributors:* Jnlin *Original artist:* Jnlin
- **File:Internet_map_1024.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg *License:* CC BY 2.5 *Contributors:* Originally from the English Wikipedia; description page is/was here. *Original artist:* The Opte Project
- **File:Lock-green.svg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/6/65/Lock-green.svg> *License:* CC0 *Contributors:* en:File:Free-to-read_lock_75.svg *Original artist:* User:Trappist the monk

10.3 Content license

- Creative Commons Attribution-Share Alike 3.0