Distributed Hash Tables

Justin Parsons and Jack Winslow

Abstract

With the increasing popularity of distributed systems and applications, distributed hash tables have become more relevant given their ability to manage widespread partitioned resources. By implementing a lookup service to work in coordination with a modified hash table, data search can become highly efficient and fault tolerant. This is ideal for building applications on top of peer to peer networks, which fare better under high traffic in comparison to the traditional client-server model. In this paper we identify the processes behind distributed hash tables, as well as highlight what makes them so important. We then detail current applications of distributed hash tables, and explore future implementations that could benefit modern technology.

Distributed Process

Distributed hash tables are a data structure optimized to efficiently store data across numerous connected nodes. Their concept is similar to that of a local hash table, but altered to support lookup in a distributed environment. Instead of getting keys by hashing a name, distributed hash tables hash the value of the resource itself. They also implement an underlying lookup service that finds the correct node storing the requested data. Given that no single node or server is relied upon for serving information to the rest of the system, this allows them to be highly scalable, naturally balancing load as new nodes join the network.

A nontrivial problem that arises with distributed data storage is how a node with the requested data can be found efficiently. This is called the Lookup Problem, and has been addressed with a variety of approaches attempting to solve this with maximum efficiency.

1. Centralized lookup involves a shared central source that directs others to the node storing some requested data. This provides nodes in the system with fast access to any resource being stored, but fault tolerance is extremely low given that the uptime of one source is a necessity to the operation of the system as a whole. In large systems using this approach, this central source could also see issues processing concurrent requests in times of high traffic, which negates any benefits the distributed hash table provides.
2. Flooded lookup maintains a system of equilibrium between all nodes, where there is no shared central source. To find data, the requesting node queries all nodes that it is directly connected to, and then each of those nodes follow suit in a trickle-down manner. By the end of the process, each node in the system has been queried. This process is extremely inefficient as a multitude of unnecessary queries are made, even though equality is maintained in the workload of each node.

With both centralized and flooded lookup each having their benefits and drawbacks, a better method for data lookup must be implemented to make the use of distributed hash tables advantageous. Routed lookup, or Chord, has become the accepted method of executing this. Chord is a protocol and algorithm used to partition data across numerous nodes in a way that makes searching highly efficient, scalable, and fault tolerant. To achieve this, it utilizes two trivial processes: consistent hashing and finger tables.

Consistent hashing is a technique used to position keys and nodes along an imaginary circle (hash ring) based on their hashed identifiers. Each node points to its successor, creating a

circularly linked list with keys falling in between the nodes along the circle. In practice, keys are stored in the successive node, so that any keys with identifiers less than the next node's identifier but greater than the last node's identifier are stored in the next node. This makes the location of keys predictable in a large system and allows for dynamic changes without disrupting the whole process of data partitioning. The downfall to this method is that searching for keys within this circular model takes linear time, which is not ideal for a data structure that is utilized for its quick lookup. To improve this lookup time, Chord implements finger tables alongside consistent hashing.

Finger tables are a list of node addresses in the hash ring that are strategically stored to improve traversal time. Any given node will store the node halfway across the ring from it, one quarter of the ring away from it, one eighth, and so on. This gives each node multiple routes to travel during traversal, and in using the most efficient route a node can find the requested data in logarithmic time. With Chord's utilization of consistent hashing and finger tables, distributed hash tables have a lookup service with efficient time and high fault tolerance, making them an ideal option for distributed data storage.

Importance

In most modern applications, the client-server model is used as the primary infrastructure for connecting devices in a system. Fault tolerance and load balancing has been improved at scale for these systems by allocating additional servers and geographically routing clients to nearby servers. While this is a good approach for maintaining uptime and smooth functionality, it is costly to allocate enough resources to support large applications and their user base. A more organic approach to support large applications is to implement a peer to peer model. These

systems are better suited for scale and fault tolerance as each node contributes computing power and there are multiple paths to fetch resources throughout the network. The issue with this model, however, is the difficulties experienced in managing resources at scale, which have contributed to the lack of adoption of peer to peer systems.

Distributed hash tables and the benefits they provide ease these management difficulties and allow for peer to peer systems to function efficiently. This opens up a range of opportunities for infrastructure development, making the peer to peer model adoptable for any application that needs to balance load and tolerate fault in highly utilized systems. In contrast to client-server based systems, peer to peer systems thrive as new nodes are added as it organically increases its computing power and data distribution. It is also much more cost effective, as new servers don't need to be allocated in the way that client-server systems require at scale. Some systems have already begun to use these structures in building efficient peer to peer systems, while others are currently being built with the hopes of changing how the internet works at a network level.

Applications

Distributed hash tables have found significant use as a decentralized lookup index for peer-to-peer file sharing. The widespread success of P2P file sharing started with the launch of Napster, a music streaming site that used a centralized lookup server to track files. Because of its centralized nature, the company was liable for copyright infringement resulting in lawsuits that took down the service. In light of this, Gnutella was born using a flooded lookup that took care of liability but was slow and computationally expensive. BitTorrent has become more relevant, which uses a distributed hash table to allow for much greater efficiencies. The Mainline DHT used by BitTorrent is O(*log* n) rather than a flooded lookup of O(n). This greater speed allows for

the protocol to maintain usefulness at scale and the implementation of features such as file partitioning for more efficiency.

IPFS is an evolution in the decentralized space inspired by BitTorrent that aims to be a P2P mutable file system. It uses the BitSwap protocol to distribute data in chunks to peers which is tracked by a distributed hash table shared among all of them. The chunks represent a file in a Merkle DAG structure similar to how files are tracked in git; the structure allows for version control, tamper resistance, and deduplication (since chunks shared by multiple files can be reused). IPFS has grown to become a focal point of the web3 revolution, popularly being used as the storage system for smart contracts and decentralized apps such as NFTs. It also can work as the backend for websites to provide the benefit that hosting is done by anyone interested in keeping the resources public, not just the website owner. This all goes to show that DHTs are essential to providing the necessary file lookup capabilities in a way that scales with the growth of the protocol and is technology that has persisted with evolutions in peer to peer distribution.

Conclusion

Because of the distributed hash tables utilization of Chord, it is a highly efficient system for lookup. This efficiency makes it viable as a self-scaling solution to manage peer-to-peer systems. It has been greatly used in P2P file sharing such as BitTorrent and IPFS. In recent years, there has been a push towards decentralization and P2P computing, especially given the emergence of web3 as a concept, and new technologies like webRTC which allows for easy implementation of widely supported browser-based peer connections. Distributed hash tables are a necessary technology to ensure decentralized and peer-to-peer systems work at scale.

Bibliography

[1] V. Iancu and I. Ignat, “A peer-to-peer consensus algorithm to enable storage reliability for a decentralized distributed database,” in *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, Cluj-Napoca, Romania, May 2010, pp. 1–6. doi: 10.1109/AQTR.2010.5520830.

[2] “A Tutorial on Bittorrent, Freenet and Gnutella Protocols.” http://medianet.kent.edu/surveys/IAD06S-P2PArchitectures-chibuike/P2P%20App.%20Survey%20Paper.htm (accessed Nov. 04, 2022).

[3] I. Stoica *et al.*, “Chord: a scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Trans. Networking*, vol. 11, no. 1, pp. 17–32, Feb. 2003, doi: 10.1109/TNET.2002.808407.

[4] H. Zhang, Y. Wen, H. Xie, and N. Yu, “DHT Applications,” in *Distributed Hash Table: Theory, Platforms and Applications*, H. Zhang, Y. Wen, H. Xie, and N. Yu, Eds. New York, NY: Springer, 2013, pp. 39–55. doi: 10.1007/978-1-4614-9008-1_4.

[5] H. Zhang, Y. Wen, H. Xie, and N. Yu, “DHT Theory,” in *Distributed Hash Table: Theory, Platforms and Applications*, H. Zhang, Y. Wen, H. Xie, and N. Yu, Eds. New York, NY: Springer, 2013, pp. 5–22. doi: 10.1007/978-1-4614-9008-1_2.

[6] C. Dubnicki, C. Ungureanu, and W. Kilian, “FPN: a distributed hash table for commercial applications,” in *Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, 2004.*, Jun. 2004, pp. 120–128. doi: 10.1109/HPDC.2004.1323509.

[7] H. Zhang, Y. Wen, H. Xie, and N. Yu, “Introduction,” in *Distributed Hash Table: Theory, Platforms and Applications*, H. Zhang, Y. Wen, H. Xie, and N. Yu, Eds. New York, NY: Springer, 2013, pp. 1–3. doi: 10.1007/978-1-4614-9008-1_1.

[8] J. Benet, “IPFS - Content Addressed, Versioned, P2P File System,” p. 11.

[9] “It’s time for the Permanent Web.” https://blog.neocities.org/its-time-for-the-permanent-web.html (accessed Nov. 04, 2022).

[10] P. Maymounkov and D. Mazières, “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric,” in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, Berlin, Heidelberg, Mar. 2002, pp. 53–65.

[11] L. Wang and J. Kangasharju, “Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT,” p. 10.

[12] “Understanding Merkle Trees. the quintessence of Git, Bitcoin and… | by Kumar Swapnil | Geek Culture | Medium.” https://medium.com/geekculture/understanding-merkle-trees-f48732772199 (accessed Nov. 04, 2022).