Distributed Hash Table

Final Project for CSCI 6421

Team Members

Guangyuan Shen (G48268794)
Xiaoyu Shen (G33150730)
Xuzheng Lu (G34363475)
Zetian Zheng (G36558368)

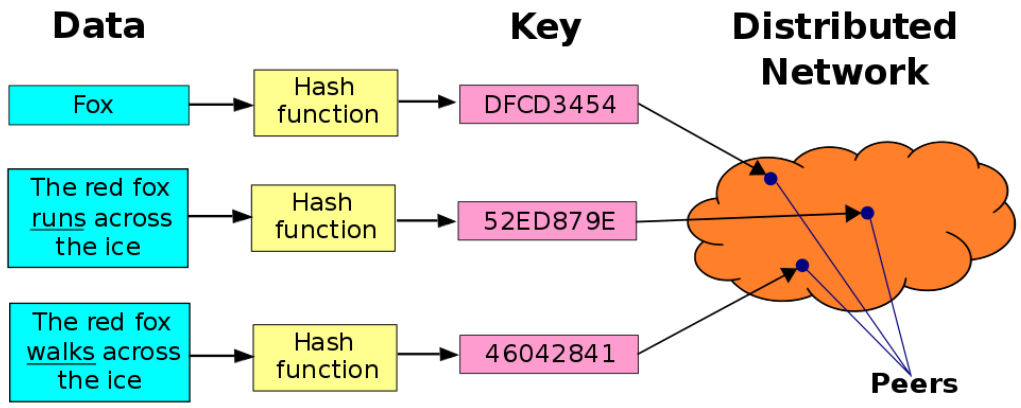
Introduction

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers.(Wikipedia: peer to peer)[https://www.wikiwand.com/en/Peer-to-peer]

In this kind of distributed system, "peer to peer" means that the function of each node is equal, and each node is connected with others via the internet. In this network, files are transported between nodes. There are many features for peer to peer system, for example, anonymity, hierarchical naming, Lookup, data storage…***the most important feature is the efficient location of data items[2]***

There are two kinds of p2p systems, unstructured networks, and structured networks. In unstructured networks, nodes randomly form the connection to other nodes. For example, Gossip, Kazaa, Gnutella are unstructured protocols. In structured networks, the connection between nodes is followed by a specific topology provided by protocols, and nodes can efficiently find the resources that they want.**(Wikipedia: peer to peer)[https://www.wikiwand.com/en/Peer-to-peer]**

Distributed hash table(DHT) is a protocol for structured p2p networks used. (Wikipedia: Distributed hash table).



The network maintains a huge file index hash table, divided and stored on each node of the network according to certain rules, with entries in the form of (key, value). Usually, the key is the hash value of the file, and the value is the IP address where the file is stored. Given the key, the value/address stored in the node can be efficiently found and returned to the query node.

Before I start, I would like to introduce some measurements for efficiency in a peer to peer system from part of the IRIS projects(http://project-iris.net/):

1. **degree:the number of neighbors with which a node must maintain continuous contact;**
2. **hop count: the number of hops needed to get a message from any source to any destination;**
3. **The maintenance overhead: how often messages are passed between nodes and neighbors to maintain coherence as nodes join and depart;**

To evaluate the performance of a peer-to-peer system, we could refer to the measurements in IRIS projects (http://project-iris.net):

1. **Degree: the number of neighbors that the node must keep communicating with;**
2. **Hop count: the number of hops required for a message sent from any source to any destination in the system;**
3. **Maintenance cost: the frequency of message transfer between nodes to maintain the consistency when nodes join and leave;**

Challenges

1. Load Balance

In a peer-to-peer network, load balancing is critical to the efficient operation of DHT. If the data of all nodes is not well balanced, we can encounter a situation where a node is

overused while other nodes are rarely accessed, but need to ping neighboring nodes repeatedly to ensure that the node is online or not. I**n David R. Karger and Matthias Ruhl's Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems (2006)[4],** they provide two protocols to enable load balancing. The first system allows data to be stored relatively randomly in the address space by increasing the 1/n factor, making it difficult for malicious nodes to attack. The second protocol removes the restriction on each node, allowing data to flow freely within each node. According to the entropy principle, objects tend to increase in the direction of entropy, so the distribution within each node will be more uniform, making the load balanced.

2. Fault tolerance

In a distributed hash table, it is possible to encounter a node failure or node exit at any time, so we need to build a suitable structure to improve fault tolerance. **In Moni Naor and Udi Wieder's A Simple Fault-Tolerant Distributed Hash Table (2003)[5]**, they mentioned that by building an overlapping DHT, nodes storing the same data are connected to form groups. When accessing one node, it is also possible to access other nodes storing the same node. And when one node is offline, other nodes can also form a group. The routing table needs to be updated as well. Due to the logarithmic degree and logarithmic dilation approach, this will also allow the time complexity of the search to be within log(n) and can effectively reduce the dilation.

3. Scalability

In a system with n number of nodes, for a constant k, θ(logn) hops is optimal. **If we want to keep a high degree of fault tolerance, a node must maintain O(logn) neighbors[1].** This is a very important concept to practice DHT: for a p2p system, nodes should be autonomous and decentralized, each node also needs "routing" information to other nodes, so they may lose connection with other nodes if nodes leave and join the system. ***Some previous work assumes that the node can be aware of most of the other nodes in the system*[2],** this approach is face to the problem of scalability. In this challenging topic, we also need to consider about:

- repartition the affected keys on the existing node;
- reorganize the neighbor nodes;
- to connect new nodes to DHT through a guiding mechanism.

Approaches

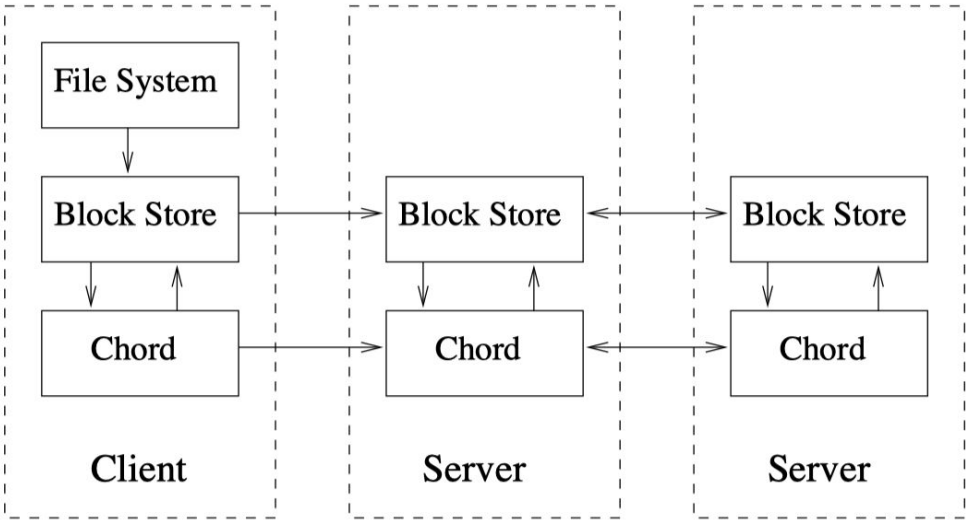
1. CAN DHT

Sylvia Paul Ratnasamy et al. proposed the Content-Addressable Network (CAN DHT) in 2001. CAN was designed as a scalable, fault-tolerant, and self-organizing distributed hash table. It is similar to a multi-dimensional Cartesian coordinate space, and each node in the network can be identified in the space. Each CAN node maintains a routing table that saves each neighbor's IP address and the coordinate area in the virtual space. The node can route the message to the target area in the coordinate space and organize the entire network.

Ratnasamy, Sylvia, et al. "A scalable content-addressable network." *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. 2001.

2. Chord DHT

Ion Stoica et al. proposed Chord DHT in 2001. Chord covers the entire network by stringing the nodes' IDs from small to large to form a ring, and each data is stored at the backward node closest to the data key. Chord uses the similar dichotomy search method, a non-linear search algorithm, and the convergence speed can be very fast. The routing complexity can be reduced to the logarithm of the number of nodes.
The structure of example chord-based system as shown in the picture:



Stoica, Ion, et al. "Chord: A scalable peer-to-peer lookup service for internet applications." *ACM SIGCOMM Computer Communication Review* 31.4 (2001): 149-160.

3. Pastry DHT

Antony Rowstron and Peter Drusche proposed Pastry DHT in 2001. Like Chord, Pastry also uses a ring network topology, concatenating each node's hash identifiers into a ring. The difference from Chord is that Pastry introduces the idea of grading, and it does not directly use node hash values to construct a ring but only take the first 'a' bit of the node's hash value and the back 'b' bit as the leaf node of a node on the ring. The advantage of this is that it greatly reduces the ring's space size and can reduce the routing time. Another difference is that Pastry's data is stored on the node closest to the key-value, not just in the backward node.

Rowstron, Antony, and Peter Druschel. "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems." *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, Berlin, Heidelberg, 2001.

4. Tapestry DHT

Ben Y. Zhao et al. proposed Tapestry DHT in 2004. In the Tapestry network, each node will assign itself a globally unique node ID, and at the same time, there is a unique root node ID in the entire system. All nodes form a tree topology in the network, and data is saved on the node closest to the key value of the data.

Zhao, Ben Y., et al. "Tapestry: A resilient global-scale overlay for service deployment." *IEEE Journal on selected areas in communications* 22.1 (2004): 41-53.

5. Kademlia DHT

Petar P. Maymounkov and David Mazieresproposed Kademlia DHT in 2002. It does XOR operation on any two nodes' IDs, and the value obtained is used as the distance measure between these two nodes. Through this distance measurement, a brand-new binary tree network topology is established. Compared with the previous four methods, the routing query speed is much improved. In the Kademlia network, the distance between a specific node and any other node is unique in the entire network. At the same time, when saving data, the target data will be stored in the N nodes closest to the data key. Even though there are N-1 nodes offline, the Kademlia network can provide the data as usual.

Maymounkov, Petar, and David Mazieres. "Kademlia: A peer-to-peer information system based on the xor metric." *International Workshop on Peer-to-Peer Systems*. Springer, Berlin, Heidelberg, 2002.

6. S/Kademlia DHT

In 2007, Ingmar Baumgart and Sebastian Mies proposed an improved method based on Kademlia DHT from the perspective of safety and anti-interference - S/Kademlia DHT. S/Kademlia can resist solar eclipse attacks, witch attacks, customer churn attacks, hostile routing attacks, and denial of service attacks. Besides, in terms of node search, S/Kademlia proposes a method of searching through disjoint paths, which further reduces the search time complexity.

Baumgart, Ingmar, and Sebastian Mies. "S/kademlia: A practicable approach towards secure key-based routing." *2007 International Conference on Parallel and Distributed Systems*. IEEE, 2007.

7. IPFS

IPFS is mainly based on the S/Kademlia algorithm and combines the advantages of Chord as well. Compared to the design of BitTorrent, it is more secure and reliable. IPFS can also use storage space more efficiently, this is because that the smallest unit of IPFS is block and IPFS has less redundant data. Juan Benet proposed that IPFS combines a distributed hash table, an incentivized block exchange, and a self-certifying namespace.

Benet, Juan. "Ipfs-content addressed, versioned, p2p file system." *arXiv preprint arXiv:1407.3561* (2014).

Project Proposal

In a distributed hash table, there is a load balancing problem, such as the data allocated to each part is unbalanced, so that there is too much data in one node, which will take a lot of time to transfer if you can use the Simple Efficient Load Balancing by David R. Karger and Matthias Ruhl. **The two protocols mentioned in Algorithms for Peer-to-Peer Systems (2006)[4]** allow for data load balancing. As for the fault-tolerant mechanism, which is needed both when a node makes an error or exits, **we can use the overlapping DHT proposed by Moni Naor and Udi Wieder[5]**, which can handle node exits correctly when a node makes an error. When the system stability problem is effectively solved by using load balancing and overlapping DHT.

In addition, when constructing a hash table, you need to choose the appropriate data structure, and when choosing a chrod, you need to concatenate the data to form a circular table, by also recording the routing table at each node to indicate the location of the next hop. In an ideal environment, we usually assume that nodes will be exited or added one by one, while in reality there are usually simultaneous exits and additions of nodes, which must not only take care of processing transfers, but also safely add new nodes, so this time after processing each node a single check of the node before processing new operations.

Reference

1. Kaashoek, M. Frans, and David R. Karger. "Koorde: A simple degree-optimal distributed hash table." International Workshop on Peer-to-Peer Systems. Springer, Berlin, Heidelberg, 2003.
2. Stoica, Ion, et al. "Chord: a scalable peer-to-peer lookup protocol for internet applications." IEEE/ACM Transactions on networking 11.1 (2003): 17-32.
3. Zave, Pamela. "Reasoning about identifier spaces: How to make chord correct." IEEE Transactions on Software Engineering 43.12 (2017): 1144-1156.
4. Karger, D., & Ruhl, M. (2006). Simple Efficient Load-Balancing Algorithms for Peer-to-Peer Systems. Theory of Computing Systems, 39(6), 787–804. https://doi.org/10.1007/s00224-006-1246-6
5. Moni Naor, & Udi Wieder. (2003). A Simple Fault Tolerant Distributed Hash Table. International Workshop on Peer-to-peer Systems. Springer Berlin Heidelberg.
6. Benet, Juan. "Ipfs-content addressed, versioned, p2p file system." *arXiv preprint arXiv:1407.3561* (2014).
7.