Chord – Scalable p2p Lookup Service

Zaid Albirawi

Abstract

The lookup service is the core of any peer-to-peer application. Chord provides a peer-to-peer lookup protocol that is scalable and that is able to handle the constant change in peer-to-peer networks.

What makes Chord more attractive than other lookup protocols is its simplicity, provable correctness, and provable performance.

1. Introduction

Peer-to-peer applications are used to share data between the nodes of a decentralized network. When looking up information in a decentralized network, each node of the network needs to know if the information that they are looking for exists in the network. Furthermore, they need a space, and time efficient solution to solve the problem. The naïve solution of this problem is to allow each node to store information about each of the other nodes in the network as well as what data they have. This solution will achieve optimal time complexity of $O(1)$, however, it will require each node in the network to store information about each of the other nodes in the network, $N-1$ nodes, as well as information about what data each of these nodes contain. Therefore, the space complexity of such solution is $O(N^2)$. In addition, peer-to-peer networks experience continuous changes as nodes join and leave the network regularly. With the naïve solution, every time a node joins or leaves the network each node must update their data to reflect the change. Hence, the naïve solution does achieve optimal time complexity, however, it is not practical.

If the naïve solution to this problem is not practical, why go through the trouble of designing a lookup protocol for decentralized network?

Decentralized networks have many advantages. Since every node in the networks acts a server and a host there is no single point of failure. Decentralized networks do not need special hardware to run as mentioned above every node in the network is the server and the client. Furthermore, decentralization does not allow for a governing body to control the network which allows for neutrality in the network.

The core of any peer-to-peer application is its lookup service. Therefore, Chord can be used in any peer-to-peer applications from torrents and filesharing to blockchain applications.

The paper, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, by Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan [21] explains how the Chord protocol solves the lookup problem while providing a scalable and efficient solution.

The rest of this paper is structured as follows. Section 2 compares Chord to related work. Sections 3 describes the algorithm and its main functions. Section 4 provides the proof of correctness for algorithm. Sections 5 analyzes the algorithm performance. Section 6 discusses potential algorithm improvements. Section 7 explains the challenges faced during the implementation of the algorithm as well as data structures used in the implementation. Section 8 is the conclusion of the report. Section 9 provides instructions on how to run the simulator. Section 10 lists the references.

2. Related Work

There are many papers that tackle the lookup problem of decentralized peer-to-peer systems. This is how Chord differs from other systems:

1. Chord's ability to determine if a document exists in the network sets it apart from other systems like Freenet [5]. Chord achieves this property by assigning documents to specific nodes based on the values they receive from consistent hashing [13]. Additionally, consistent hashing allows Chord to balance the documents through the network and distribute those documents equally on the network nodes. Systems like Freenet and Ohaha do not assign documents to nodes. Instead they perform a search through cached copies which does not allow them to accurately determine if a document exists in the system.

2. The DNS [15] lookup protocol allows users to map domain names to IP addresses. Chord has the ability to perform the same functionality, however, it is less restrictive. The DNS protocol requires specific naming structure and specialized servers while the Chord protocol does not.

3. The CAN protocol is a more complicated protocol than Chord. Instead of using consistent hashing it uses a d-dimensional Cartesian coordinate space to implement a distributed hash table that maps keys onto values [20]. That means that each node will have their own hash table which will allow it to perform lookups on the network. Since the d value is a constant then when a network is constructed a d value must be chosen for efficient lookup times. Therefore, CAN might not process lookups optimally in constantly changing network while Chord will.

3. Description

The Chord protocol allows the user to perform lookup operation in the peer-to-peer network. The protocol will use consistent hashing to assignment documents to node IDs. Each node in the network will have a unique ID ranging from 0 to $2^m - 1$ where 2^m is the maximum number of documents and nodes that the network can contain.

The idea of peer-to-peer networks is to share information between the peers of the network. Each node that joins the network will have its own set of documents that it can share. However, it might not be the node that will share those documents. Once the node joins the network the documents that it wants to share are assigned keys using constant hashing and are sent to the node that is responsible to serve those documents to the network.

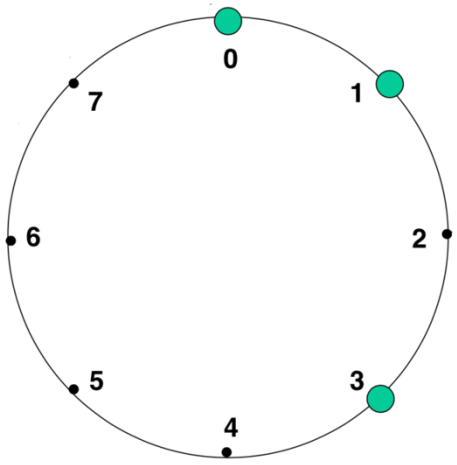

Figure 1: Peer-to-peer network with 3 nodes at locations 0, 1, and 3.

Each node in the network is assigned a range of keys that have an ID smaller or equal to the ID of that node and higher than the ID of that node's predecessor. In Figure 1, node 0 will be assigned keys between 4 – 0, node 1 will be assigned the key 1, and node 3 will be assigned keys 2 and 3. Of course not every key in the range will exist in the network. Therefore, since every key has to be assigned to a node, then, if that key does not exist under that node then the key does not exist in the network.

To perform a lookup operation each node needs to contain information about other nodes in the network which will allow it to navigate the network and find the document required. Chord achieves that by using finger tables. The finger tables allow each node to travel across the network. Each finger table will at most have $m - 1$ entries. The entries are the addresses of the successor nodes that are 2^i away from that node, where x ranges from 0 to $m - 1$. If node $2^x + ID$ does not exist then the address in the finger table is set to the node that would be its successor.

```
build_finger_tables(nodes, m):
for n in nodes:
  s = n.succ
  for i = 0 to m - 1
    if (n.id + 2^i <= s.id)
      n.fingers.add(s.add)

    while (n.id + 2^i <= s.id)
      ++i
    s = n.succ
```

As soon as a node joins the network, they build their finger tables and send the documents they're sharing to the node responsible for these documents. Once this initialization step is over the node will be able to perform look up operations.

The look up operation is simple. Once the node obtains the key of the document it is looking for, the node checks if its ID is larger or equal to the key. If it is, then this is the node that contains the documents. Therefore, search the node keys, if the key exists then return the document. Otherwise, compare the key against the keys of the addresses in the fingers table. If the successor's ID is larger or equal to the key then pass the query on to the successor, else, find the successor node that has largest ID that is smaller or equal to the key to pass the query onto that successor.

```
find(id, addr, key, keys, fngs):
if id >= key
  if keys contain key
    return address
   else return FALSE

succ = fngs[0]
if succ.id >= key
  msg <- {succ.addr, 'Q', key}

while succ.next.id <= key
  succ = succ.next
  msg <- {succ->addr, 'Q', key}

while(TRUE)
  if msg != null
    send(msg)

  if msg.addr != addr
    exit()

msg <- receive()

if msg != null

  from <- msg.addr
  type <- msg.type
  key <- msg.key

  if type == 'Q'

    if id >= key
      if keys contain key
        msg <- {from, addr}
      else
        msg <- {from, FALSE}

    successor = fngs [0]
    if succ.id >= key
      msg <- {'Q', key, succ}

    while succ.next.id <= key
        succ = succ.next
        msg <- {'Q', key, succ}

 else return msg.addr
```

Finally, the Chord protocol handles continues node arrivals and departures from the network by moving the keys of the departing nodes to their successors and the keys from the successors of the arriving nodes to the arriving nodes. The arriving nodes will receive all the keys that are equal or less than the ID of those nodes from their successors. Upon any arrivel or departure the finger tables are also rebuilt for all nodes in the network.

4. Proof of Correctness

Every node in the network has the address of the its successor as its first entry in its finger table. Additionally, every key is assigned to a node in the network. Therefore, since very node is reachable from at least one node and since every key is assigned to one node then any query for a key will result in either finding the key or proving that it does not exist in the network.

5. Performance Analysis

The time complexity for the lookup algorithm is $O(\log N)$ and the space complexity is also $O(\log N)$. Since every node in the network contains at most $m - 1$ fingers containing the address of $2^x + ID$ and since the network can only contain 2^m nodes/keys in the worst-case scenario the lookup query will have to send $m - 1$ messages and travel through $m - 1$ nodes. Finally, since at most

$$N = 2^m$$

then,

$$\log N = \log 2^m$$

and,

$$\log N = m$$

therefore, the time complexity is $O(\log N)$ and so is the space complexity as we each node stores $m - 1$ fingers.

6. Improvements

Chord is a great protocol that is difficult to improve. However, in this section I will discuss what I believe could be improved in the Chord protocol.

In larger networks instead of assigning documents to nodes, I believe that it could be beneficial to assign documents to node clusters. This will help reducing the effects of continuous change in the network caused by constant arrivals and departures of nodes. In these clusters each node will mirror the other.

This modification will provide redundancy, load balancing, and lessen the effects of changes in the network. If any of the nodes inside a cluster crash or leaves the network there will be $k - 1$ other nodes that will contain the same information therefore skipping the process of moving keys from a node to another. Load balancing will be handled by the cluster which will route queries to the different nodes inside of it allowing workload to be shared among multiple nodes. The modification will also allow the finger tables to be more consistent as they will only need to change if new cluster are added or removed. New nodes will only need to contact one of their cluster neighbors to build their finger tables and receive their data.

Such modification will only be useful when the network contains a large number of nodes. Otherwise if each cluster contains a small number of nodes then it will provide no improvement over the current Chord protocol.

7. Implementation

The algorithm is implemented in the form of a simulator. The simulator allows the user to interact with the network and perform four different tasks.

1. Perform a lookup in the network,
2. add a new node,
3. terminate an existing node,
4. and finally, crash an existing node.

The program simulates the results of a Chord protocol and provides output the describes the network state after any of those actions.

The simulator starts by initializing a network structure that contains all the information in the network, the maximum size of the network, the number of nodes, and the keys in the network. That information is populated from an input file that is structured as follows:

```
m = 4
n = 6
k = 10
1
3
4
5
6
8
9
11
13
15
0
2
6
9
13
14
```

The first three entry m, n, and k define the value m of the network, the number of nodes in the network and then number of keys respectively. The next k numbers are the values of the keys in the network and n numbers that follow are the identifiers for the network nodes.

Notes:

- All numbers must be entered in an ascending order,
- There must be at least 2 nodes and 2 keys in the network at all times for it to function as expected.

Once the network structure is initialized the simulator assigns keys and builds finger tables for all the nodes in the network.

Following the initialization phase the system will be ready for the user interactions

and requests commands through the command line to perform the previously mentioned available actions on the network.

a. Challenges

The Chord protocol is popular for its simplicity, therefore, there weren't many challenges in the implementation phase that were worth noting other than writing the logic to determine whether a node value is higher than another node's value or a key value once the values loop in the logical ring.

To explain this, I will use node 0 and keys 14, 15, and 0, in a network where $m = 4$ the highest value node is 13. The keys 14, 15, and 0 should be assigned to node 0. However, to determine if a key belongs to a node we test if the key is smaller or equal to the node. This was resolved adding the maximum value in the network to the value of the node id, 2^m, whenever that case existed.

b. Data Structures

The simulator uses a few data structures to simplify the implementation of the algorithm.

Firstly, the simulator uses a network structure to store all the information about the network.

```
typedef struct
{
  int m;
  linked_list_t *processors, *keys;
} network_t;
```

The variables processors and keys hold the values of all the networks processors and keys in a linked list.

Additionally, the simulator makes use a processor structure that maintains the values for each node in the network.

```
typedef struct
{
  short id;
  int n, active;
  struct node *keys;
  linked_list_t *fingers;
} processor_t;
```

The id value holds the node's identifier value. The integers n and active store the number of keys the node is responsible for and whether the node is active or crashed. The keys pointer points to the first key that node is responsible for. It is a node of a linked list, therefore, when used along with the n value you can determine all the keys that process is responsible for. Finally, the fingers linked list contains the processors fingers.

8. Conclusion

To conclude, the Chord protocol is scalable and efficient even in a continuously changing network. When applied to larger networks it can be improved by clustering its nodes to allow for redundancy, load balancing and more efficient operations in a constantly changing network.

9. Appendix

a. Compile**:** The simulator can be compiled by navigating to the directory of the application and executing the make command.

b. Run: Once the application is complied, execute the command, ./simulator.

c. Use: Once executed the program will prompt the user with a set of actions that they can perform on the network. The user will choose the number of the action that they want to perform and entry any other data required for the program to execute that action (ex. Processor ID or key value.)

10.References

[1] ANDERSEN, D. Resilient overlay networks. Master's thesis, Department of EECS, MIT, May 2001. http://nms.lcs.mit.edu/projects/ron/.

[2] BAKKER, A., AMADE, E., BALLINTIJN, G., KUZ, I., VERKAIK, P., VAN DER WIJK, I., VAN STEEN, M., AND TANENBAUM., A. The Globe distribution network. In *Proc. 2000 USENIX Annual Conf. (FREENIX Track)* (San Diego, CA, June 2000), pp. 141–152.

[3] CHEN, Y., EDLER, J., GOLDBERG, A., GOTTLIEB, A., SOBTI, S., AND YIANILOS, P. A prototype implementation of archival intermemory. In *Proceedings of the 4th ACM Conference on Digital libraries* (Berkeley, CA, Aug. 1999), pp. 28–37.

[4] CLARKE, I. A distributed decentralised information storage and retrieval system. Master's thesis, University of Edinburgh, 1999.

[5] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability* (Berkeley, California, June 2000). http://freenet.sourceforge.net.

[6] DABEK, F., BRUNSKILL, E., KAASHOEK, M. F., KARGER, D., MORRIS, R., STOICA, I., AND BALAKRISHNAN, H. Building peer-to-peer systems with Chord, a distributed location service. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)* (Elmau/Oberbayern, Germany, May 2001), pp. 71–76.

[7] DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)* (To appear; Banff, Canada, Oct. 2001).

[8] DRUSCHEL, P., AND ROWSTRON, A. Past: Persistent and anonymous storage in a peer-to-peer networking environment. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS 2001)* (Elmau/Oberbayern, Germany, May 2001), pp. 65–70.

[9] FIPS 180-1. *Secure Hash Standard*. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, Apr. 1995.

[10] Gnutella. http://gnutella.wego.com/.

[11] KARGER, D., LEHMAN, E., LEIGHTON, F., LEVINE, M., LEWIN, D., AND PANIGRAHY, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (El Paso, TX, May 1997), pp. 654–663.

[12] KUBIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., EATON, P.,

GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. OceanStore: An architecture for global-scale persistent storage. In *Proceeedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)* (Boston, MA, November 2000), pp. 190–201.

[13] LEWIN, D. Consistent hashing and random trees: Algorithms for caching in distributed networks. Master's thesis, Department of EECS, MIT, 1998. Available at the MIT Library, http://thesis.mit.edu/.

[14] LI, J., JANNOTTI, J., DE COUTO, D., KARGER, D., AND MORRIS, R. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking* (Boston, Massachusetts, August 2000), pp. 120–130.

[15] MOCKAPETRIS, P., AND DUNLAP, K. J. Development of the Domain Name System. In *Proc. ACM SIGCOMM* (Stanford, CA, 1988), pp. 123–133.

[16] MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, New York, NY, 1995.

[17] Napster. http://www.napster.com/.

[18] Ohaha, Smart decentralized peer-to-peer sharing. http://www.ohaha.com/design.html.

[19] PLAXTON, C., RAJARAMAN, R., AND RICHA, A. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ACM SPAA* (Newport, Rhode Island, June 1997), pp. 311–320.

[20] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND S H E N K E R , S . A scalable content-addressable network. In *Proc. ACM SIGCOMM* (San Diego, CA, August 2001).

[21] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. Tech. Rep. TR-819, MIT LCS, March 2001. http://www.pdos.lcs.mit.edu/chord/papers/.

[22] VAN STEEN, M., HAUCK, F., BALLINTIJN, G., AND TANENBAUM, A. Algorithmic design of the Globe wide-area location service. *The Computer Journal 41*, 5 (1998), 297–310.