COP5615
10/24/2015
Pawel Cieslewski
Will Livesey

Project 3:

Implementing Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications

Introduction

Chord is a protocol and algorithm for peer to peer distributed hash tables. A standard hash table assigns a key value pair which enables the user to enter the key and retrieve the pair faster than indexing through an array. A distributed hash table, such as Chord, builds on this by storing key value pairs to different nodes, where each node is a unique computer in the system. Chord determines how keys get assigned to each node in the system and how a node retrieves the value for a certain key by locating the first node that contains the key.

One of the biggest advantages of Chord is that it is a decentralized algorithm. Rather than being reliant on a central entity, Chord distributes it is work among all the nodes in the system. An obvious benefit to this approach is the increased robustness of the system. In the standard centralized algorithm, if the central entity dies the system is worthless; whereas, in Chord if a node dies the other nodes can update their finger tables accordingly and the system will be fine.

Analysis

Chord boasts a lookup time of log(N) with high probability, where n is the number of nodes in the system. This is achieved by using the finger tables of each node to conclude the finger closest to the range you are searching for. This allows you to skip over several nodes if you know the key is located on the other side of the ring. To verify the log(N) lookup time we computed the average number of hops, where each hop is when a node forwards the message to another node in its finger table, for different size node circles. The results can be seen in table 1.

Number of Nodes	Average Number of Hops	Number of Trials
10	0.64	5
50	2.612	5
100	3.22	5
500	4.2796	5
1000	4.8754	5
5000	5.9998	5
10000	6.49136	5

Table 1. Displays the results from our program. The number of trials represents the 'numRequests' parameter

To see a better visual representation of the logarithmic pattern, table 1 was graphed below (see figure 1).

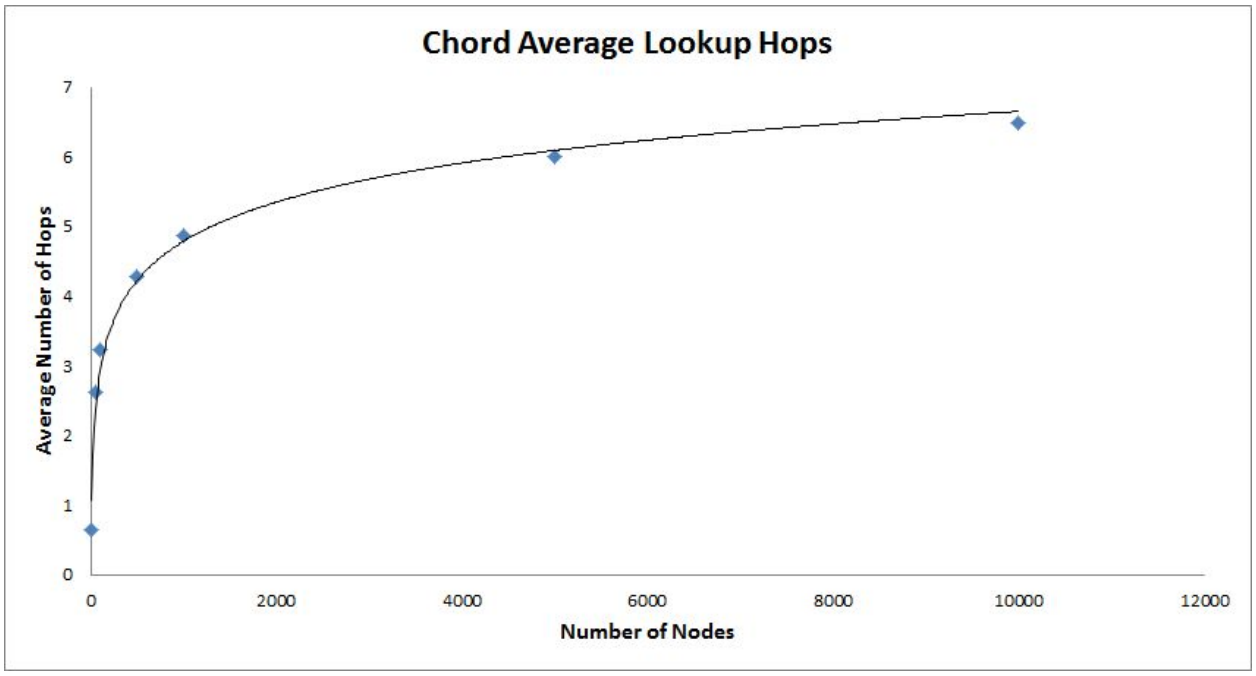

Figure 1. Displays the average number of hops versus the number of nodes in the system

Figure 1 clearly depicts a logarithmic line. Despite the number of nodes increasing at a constant rate, the average number of hops is barely increasing. This makes sense because after a queried message gets forwarded log(N) number of times it is guaranteed to be at most 2^m / N nodes away, where m is an identifier large enough so no two nodes will be hashed to the same identifier (in our case m was 62).

Chord calls for an identifier where the chances of collision is trivial. SHA-1 is commonly used for this, producing a 160-bit identifier. For the purposes of our experimentation, we used the lower 62 bits of the SHA-1 algorithm. 62 was chosen because it is able to fit in the Long primitive of Scala, and it allows the

Difficulties of Implementation and Future Work

In section 4, the paper describes a simple implementation of the chord protocol but warns the audience that there may be two problems with the described implementation.

- Simultaneous joins into the system need to be handled.
- Without modification, the implementation is not fault tolerant.

Since the assignment called for implementing section 4 of the paper, naturally our project suffers from both of these issues. If a node was to die, ring would be broken, and all of the data that was stored at that node would be lost. Fortunately, since all the nodes are being simulation on a single machine, failure was not a problem.

However, what truly became a very hard pressed issue is simultaneous joining of nodes. We noticed that in order for our chord topology to be properly generated, we needed to add a latency between new nodes joining. This is because we need the system to properly update the finger tables before another node starts updating them as well. If the nodes joined too fast, we noticed that race conditions would occur between updates sometimes leading to old information overwriting newer information. When this occurred, some nodes would get skipped over entirely in the finger tables causing difficulty with finding nodes. Since the system does not have any method of stabilization implemented, this causes a failure of the system.

Naturally, future work involves implementing the improvements to chord mentioned in the section 5 of the paper. By adding some period stabilization and redundancy, we would be able to allow for simultaneous join of nodes, and allow for some the the nodes to fail, without causing the entire system to fail.

Conclusion

It is without a doubt that the Chord protocol has proved its usefulness experimentally. We were able to see first hand how both the insertion and lookup time are Log(n). This gives the system extremely scalability in terms of number of nodes that can be in the same system since it barely

affects these times. It is certain that with some modification and appropriate redundancy Chord is an excellent backend to a large distributed system.

References

Our references include the original research paper and a wikipedia page explaining the paper.

https://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

https://en.wikipedia.org/wiki/Chord_%28peer-to-peer%29