CS441 Course Project: Chord Algorithm Akka/HTTP-based Simulator

Team:
Carlos Antonio McNulty (cmcnul3),
Abram Gorgis (agorgi2),
Priyan Sureshkumar (psures5),
Shyam Patel (spate54)

Background

This project is a simulator of a cloud overlay network that makes use of the Chord protocol, which specifies how to find the locations of keys for files. Chord assigns keys to nodes using ***consistent hashing***, providing a degree of natural load balancing. Chord is scalable, because it is not required that every node know about every other node, and the cost of a Chord lookup is always the log of the number of nodes. Our simulation stores data about movies in files distributed across the network and supports inserts and lookups.

Chord Algorithm

For comparison, our simulator consists of two Chord algorithms. In the former, a simple but slow Chord lookup algorithm is used. Lookups are implemented on a Chord ring in which each node only knows how to contact its current successor node on the identifier circle. As such, queries for a given identifier may be passed around the circle via these successor references until they encounter a node that consists of the desired identifier.

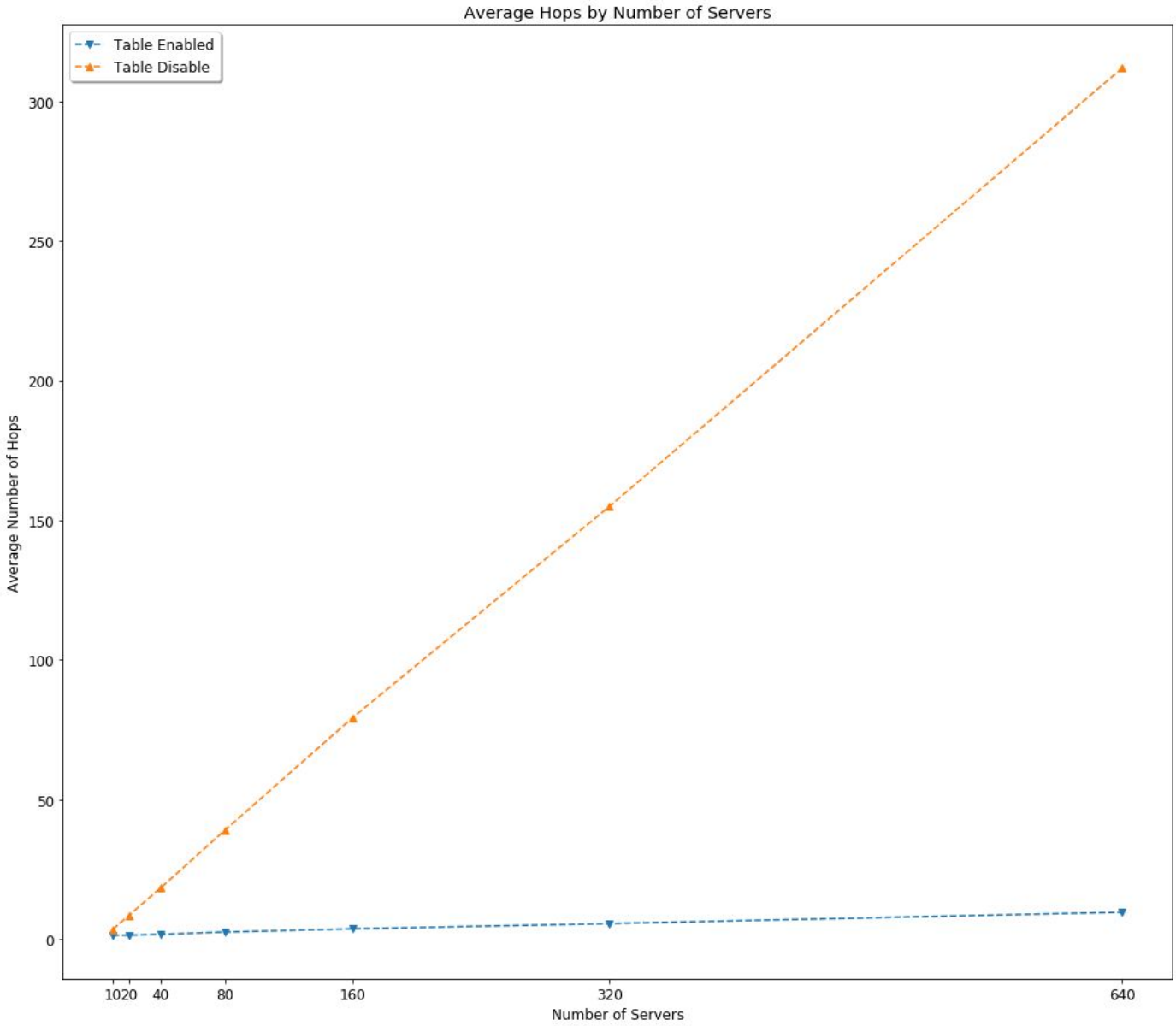
In the latter, much improved algorithm, Chord maintains additional routing information. Each node maintains a routing table known as the ***finger table***. The first finger of a node *n* is the immediate successor of *n* on the identifier circle. In this scheme, each node stores information about only a small number of other nodes, and knows more about nodes closely following it on the identifier circle than about nodes farther away. As such, given that a node's finger table generally does not contain enough information to directly determine the successor of an arbitrary key *k*, if the id to be searched does not immediately fall between *n* and its successor, node *n* searches its finger table for the node *n'* whose identifier most immediately precedes the id.

Analysis

The following are the results we observed in our simulation.

Table 1. Average Number of Hops

	10 servers	**20 servers**	**40 servers**	**80 servers**	**160 servers**	**320 servers**	**640 servers**
Simple Chord alg.	3.612	8.524	18.463	39.077	79.481	154.963	311.842
Scalable Chord alg.	1.519	1.517	1.873	2.676	3.826	5.685	9.757



In the simple Chord algorithm simulation, in which the finger table is disabled, nodes merely contact their successor nodes on the Chord ring, which results in high numbers of hops when inserting and looking up files. As can be observed in **Table 1**, when the number of servers is doubled, the average number of hops is also doubled. Therefore, the number of servers has a ***linear*** relationship with the number of hops.

In the scalable Chord algorithm simulation, however, nodes use their finger tables to find successors, which results in fewer numbers of hops when inserting and looking up files. As can be observed in **Table 1**, when the number of servers is doubled, the average number of hops increases by a factor of log of the number of servers. Therefore, the number of servers has a ***logarithmic*** relationship with the number of hops.

This was more or less expected, because the finger table avoids the need for a linear search and significantly reduces the number of successor nodes that must be found in order to locate the key. With the finger table implementation of the Chord algorithm, the number of nodes which must be contacted to find a successor in an *n*-node cloud overlay network is ***O(log n)***.