Aakhila Shaheen (shahe026@umn.edu) and Shannon Sequeira (sequeira@umn.edu)

CSCI 5105 Programming Assignment 2

Design Document

Introduction

In this course, we implement a distributed hash table (DHT) using the Chord protocol for scalable name resolution to hold book titles and their genres.

In the Chord protocol, the logical hash table is distributed into contiguous shards on different nodes, each of which is responsible for one part of the keyspace. Nodes are arranged in a ring, with neighborhood signifying that two nodes are responsible for adjacent parts of the ring. In this manner, the contiguity of the keyspace sharding is preserved.

In addition to links to predecessors and successors, nodes also maintain “fingers” that extend further out into the ring. More specifically, nodes maintain information about nodes two hops away, four hops away, and so on until the node halfway across the ring. This allows scalable key look up: by appropriately jumping across half the network, the protocol reduces the part of the keyspace to be searched by up to half. This halving of the keyspace is reminiscent of binary search, and indeed, Chord guarantees look up times that are polylogarithmic in the number of nodes in the system.

Given this overview of the protocol our system is based on, we detail the design and performance of its three components below.

Client

The client is a user’s entry into our application. The client is launched by specifying the super node’s hostname and IP address. The client launches a read-evaluate-print loop that allows the user to get and set genres via the console, as well as specify a file that will be read to populate the hash table.

The client connects to our system by first communicating with the super node for via remote procedure call (RPC). The super node responds with the address of some node in the system, which the client then contacts to submit queries into the DHT. We employ Apache Thrift to generate RPC stubs.

Aakhila Shaheen (shahe026@umn.edu) and Shannon Sequeira (sequeira@umn.edu)

Super node

The Chord protocol is silent on discovery: it assumes that new nodes become aware of other nodes by an external service. In order to facilitate nodes joining the DHT, we implement a super node that maintains state about the nodes in the ring. The super node is the first access point for the DHT, be it by a new node waiting to join, or a client requesting service.

The super node maintains a list of active nodes in the system, ward the request to the along with their assigned IDs. When a new node approaches the super node, the super node checks if any other node is currently also in the process of joining the network. To avoid the problems associated with two nodes joining at the same time, the super node only allows one node to join the system at a time and issues a message of no acknowledgement to later nodes that wish to join. On the other hand, if there is no other node joining the system, the super node assigns a unique ID in the keyspace, along with the address of its predecessor in the space. The new node can then contact its predecessor to join the ring.

When a client contacts the super node, it will return one of the nodes at random from the list of assigned nodes. Since all Chord nodes implement the same functionality, the randomness helps balance the load across the ring in case of multiple clients joining.

Chord nodes

The Chord worker nodes are the heart of the system implementation. Chord nodes handle the requests from clients, through the Chord protocol. Requests can accrue to any Chord node after network formation. Upon receipt, the node will first access its own internal table to search for the requested entry. If the request hashes into a part of the keyspace that is not under its custody, it will forward the request up the ring to another node that either contains the desired record or is closer to the node containing the desired record. This is accomplished by making use of the finger tables, which contain hops across the network for efficient routing.

Chord nodes also take part in the joining protocol, where after the super node returns the address of one of the ring members to a newly joining node, the new node contacts the other nodes in the table to update its finger table and notify the ring of its joining.

Assumptions and limitations

We assume that all components share the same underlying networked file system,hashing function. We have no other limitations on the locations of the super node, Chord nodes and

clients. During our tests, we used multiple clients and Chord nodes, across different machines that shared the same file system.

Our client has the limitation of only being able to access the DHT after all nodes have joined the ring. Finally, we assume that our nodes do not experience any failures, and consequently do not perform any error checking against this scenario.

Testing

We performed several tests on our system to verify correctness.
First, we inserted a file with book title and book genre pairs into the DHT and recalled all the pairs to verify that all pairs were correctly entered. It returned a message that all pairs were found correctly.
Second, we searched for a book that we know was not in the DHT to obtain a “book not found” error.
Third, we tested using multiple clients across different machines.
Fourth, we tested having more nodes than the maximum allowed number; additional clients were given “NACK” messages.
Fifth, we tried to have two nodes join the DHT at the same time; the later client was given a “NACK” message.
Sixth, we had the client connect to the DHT before it was completely formed; the super node refused access.
Seventh, we killed a node just to see if the client can still gracefully be intimated about the error. For this, we handled the error for internal transport exceptions and let the client know that the request could not be processed due to internal errors. To replicate this, kill one of the nodes and query for something that belongs to the dead node.
Eight, when an existing book is set again, its genre gets updated

We conclude that our testing verifies the robustness and design of our system.