CSE 486/586 Distributed Systems Project 1

Simple Key-Value Storage

Team

Pradeep Yenneti

Rikson Chukkiri Vareed

Shreyas Chickmaglur Sunil

Sujith Mohan Velliyattikuzhi

Sushrutha Sreevathsa

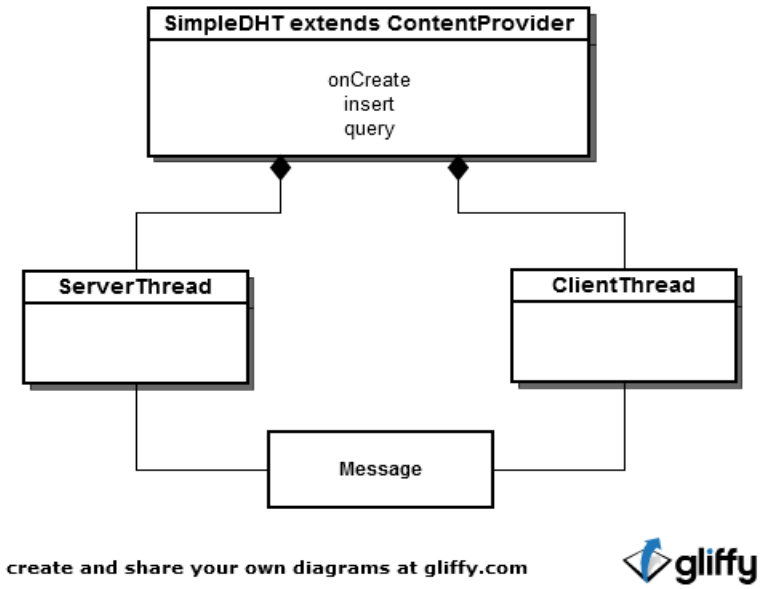
Abstract

This project involves designing a simple distributed key-value storage based on Chord. It is a simplified version of Chord and does not need implement finger tables and finger-based routing; neither does it handle node leaves/failure. The distributed hash table (DHT) is built on top of the android content provider.

Design Principles

- Keep It Simple and Stupid (KISS) – small distinct entities, each doing a specific task.
- Don't re-invent the wheel – leverage existing android interfaces.

System Architecture



Design Overview

The design is pretty simple as seen from the above diagram. The DHT (Content Provider) has a server thread and client threads which communicate by sending messages. **There are different types of message for join, insert, query etc. which simplifies the application logic**. Other noteworthy point is that the message format encapsulates the origin address so that the response can be sent back directly to the original requester rather than hopping back recursively.

Join Algorithm

If ((hash(new_node_id) > hash(predecessor) && hash(new_node_id) <= hash(current_node_id)) || hash(current_node_id) < hash(predecessor) && (hash(new_node_id) > hash(predecessor) || hash(new_node_id) <= hash(current_node_id)) :

    new_node.predecessor = current_node.predecessor

    new_node.successor = current_node_id

    current_node.predecessor.successor = new_node_id

    current_node.predecessor = new_node_id

else if (hash(current_node_id) == hash(predecessor_id) == hash(successor_id)):

    new_node.predecessor = current_node_id

    new_node.successor = current_node_id

    current_node.predecessor = new_node_id

    current_node.successor = new_node_id

else:

    forward_join(successor)

The first 'if' case handles when the new node ID is in range as per CHORD (Consistent Hashing), with the special case of modulo point (when key space crosses max to zero) being handled in the OR case.

The second 'if' case handles the special case when there is only one node in the system.

The last case handles the case where the key belongs to a different key space and thus just forwards it in the CHORD ring.

Insert | Query Algorithm

If ((hash(key) > hash(predecessor) && hash(key) <= hash(current_node_id)) || hash(current_node_id) < hash(predecessor) && (hash(key) > hash(predecessor) || hash(key) <= hash(current_node_id)) :

    // The key belongs to the local partition.

    // Therefore do the requested operation (insert/query) locally

    insert (…) | query (...)

else:

    // The key belongs to a different partition.

    // Therefore forward it in the ring

    forward (message, successor)

The above algorithm is similar to the join algorithms. The central theme is simple: if the key belongs to the local partition, do the operation locally; else forward the message in the ring.

Miscellaneous

Package: edu.buffalo.cse.cse486_586.simpledht

Eclipse project name: SimpleDHT

APK: SimpleDHT.apk