INF3200: Mandatory Assignment 1

Yasiru Rathsara Witharanage

September 25, 2021

Abstract

In an ever-growing tech-world, the necessity of storing and accessing data is vital. With the expansion of industry and users, data-stores are being required to address additional features apart from its main objective, such as scalability, replication, accessibility etc. To this end, distributed data-stores have been more useful and applicable. In this study, I have implemented a distributed hash table using chord [1] and conducted few experiments to observe how its performance differs with cluster's scale.

1 Introduction

The primary objective of this study is to build a system of distributed nodes acting as a single key-value store with the assistance of a distributed hash table 'chord' which uses a ring network of individual nodes with distributed memory each with the knowledge of only a partition of the entire network. This store exposes the following HTTP endpoints for querying and debugging purposes.

- PUT /storage/key - to store a value provided in the raw body

- GET /storage/key - to retrieve the value of a key (returned in raw body)

- GET /neighbors - to retrieve the neighbours of a node

2 Project Description

2.1 Design

Each individual node in the network is given an id based on the SHA256 hash value of its own hostname. Thus, specific instances were chosen to re-

semble and organize the ring structure such that there are no multiple instances for the same id. Fig.1 shows one of such structures with 16 nodes used in the experiments of this study along with generated node ids and hostnames.
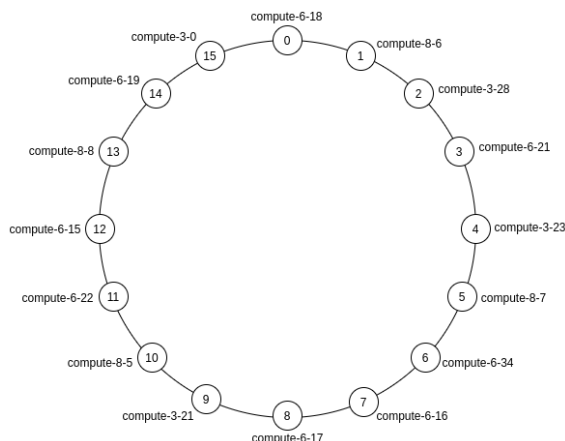

Figure 1: Fig.1 Ring structure with 16-nodes

Each node is only aware of its neighbours, i.e. predecessor and successor. These values can be configured before the execution of a node.

Whenever a query is received by a node, it initially checks whether the requested key belongs to itself and if not, it proceeds to the immediate successor. This process is iterated until the correct node for key is discovered and the corresponding response is returned along the same route.

2.2 Implementation

Golang was used for the implementation of chord with a flat-hierarchical code structure and different layers for each functionality. HTTP endpoints listed previously are implemented in http.go while node.go contains details (node id, predecessor, suc-

cessor) related with the node along with validating key function. SHA256 was used for hashing of both key and node id to match with relevant buckets. A separate HTTP client was used in neighbour.go to proceed any request with a key that does not belong to the corresponding node. All key-value entry pairs are stored in an in-memory map which is specifically designed for concurrent use.

A sample set is given below for the configuration values of parameters required by the implementation in configs.yaml but however user is allowed to provide his own desired values for service configurations. Predecessor and successor ports can be given null if they are same as the port of current node.

```
# service configs
port: 52520
max_num_of_nodes: 16
request_timeout_sec: 2
ttl_min: 180

# neighbour configs
finger_table_enabled: false
neighbour_check: false
predecessor: "compute-1-1"
predecessor_port: ""
successor: "compute-1-1"
successor_port: ""

# logger configs
colors_enabled: true
log_level: "TRACE"
file_path: true
```

2.3   Implementation

3   Evaluation

4   Results

5   Discussion

6   Conclusion