D7024E Kademlia Lab

Group 7
Elvira Forslund Widenroth
Jenny Sundström

October 2024

1 Introduction

This lab is part of the course D7024E Mobile and Distributed Computing systems at Luleå University of Technology. The objective is to produce a working Distributed Data Store using Kademlia, a protocol for facilitating Distributed Hash Tables, which stores data across multiple computers rather than one. A user should be able to store data on multiple "nodes" and later retrieve them from any node using their hashes from a Cli.

In order to implement this system, any program and container solution could be used. The system was to be developed using Agile Software Development by implementing the system in 2 Sprints. Lastly, the grading was done according to how many features had been added to the system. For this group, the mandatory features have been implemented.

1.1 System Functionality

The following are the functionalities that the system was required to have:

- **Network formation.** Nodes must be able to form networks as described in the Kademlia paper. Kademlia is a protocol for facilitating Distributed Hash Tables (DHTs). Concretely, the following aspects of the algorithm must be implemented:
  - (a) **Pinging.** Implement and use the PING message.
  - (b) **Network joining.** Given the IP address, and any other data decided, of any single node, a node must be able to join or form a network with that node.
  - (c) **Node lookup.** When part of a network, each node must be able to retrieve the contact information of any other node in the same network.
- **Object distribution.** The networks your nodes form must be able to manage the distribution, storage, and retrieval of data objects, as described in the Kademlia paper.
  - (a) **Storing objects.** When part of a network, it must be possible for any node to upload an object that will end up at the designated storing nodes.
  - (b) **Finding objects.** When part of a network with uploaded objects, it must be possible to find and download any object, as long as it is stored by at least one designated node.
- **Command line interface.** Each node must provide a command line interface through which the following commands can be executed:
  - (a) **put:** Takes a single argument, the contents of the file you are uploading, and outputs the hash of the object, if it can be uploaded successfully.
  - (b) **get:** Takes a hash as its only argument, and outputs the contents of the object and the node it was retrieved from, if it could be downloaded successfully.
  - (c) **exit:** Terminates the node.
- **Unit testing.** Test to verify core parts of the implementation work as expected by writing unit tests with a coverage of at least 50%.
- **Containerization.** Ability to spin up a network of nodes on a single machine. The network must consist of at least 50 nodes, each in its own container.

2 Implementation Details

2.1 System architecture description

The system contains the following components:

- **Network:**
  Holds: Response Channel, Connection.
  Manages: Communication between nodes over a UDP connection. It listens for incoming messages, processes them according to their type (e.g., PING, STORE, FIND_NODE, FIND_DATA), and sends appropriate responses over the Kademlia and Network channels. It also provides methods to send specific messages, such as PING or STORE, and ensures data consistency and responsiveness within the distributed network.

- **Kademlia:**
  Holds: Routing Table, Network, Data, ActionChannel
  Manages: The routing table, network communication, and stored data. Listens to incoming commands/actions via a channel and act accordingly. Key methods include LookupContact, which finds the closest contacts to a target node, LookupData, which searches for data in the network, and Store, which stores data in the DHT. It also handles network operations like sending and receiving messages and managing a shortlist of contacts to probe during node lookups.

- **Routing Table:**
  Holds: Me (which kademlia node/contact this is), buckets
  Manages:A node's contact information and organizes other nodes into buckets based on the xor distance from the local node, represented by the Me contact. The AddContact method places contacts in the appropriate bucket, while FindClosestContacts searches for the closest nodes to a target ID. The getBucketIndex function calculates which bucket a contact should be placed in based on its distance from the current node. Additionally, the table can print information about its stored contacts.

- **Contact:**
  Holds: (kademlia) id, address, distance
  Manages: The structure and behavior of Kademlia network contacts, including calculating distances between nodes and managing a list of contact candidates. It provides functionality to sort contacts by their proximity to a target node based on XOR distance. Additionally, it enables adding, swapping, and retrieving contacts from the list.

- **Bucket:**
  Holds: list (of contacts)
  Manages: A list of contacts (nodes) and operates as a part of the routing table. It includes methods to add, remove, and manage contacts, ensuring the bucket's size limit is respected by either adding new contacts or moving existing ones to the front if already present. Additionally, it provides functions to calculate distances between contacts and a target node and print contact details.

- **Kademla ID:**
  Manages: The structure and behavior of a KademliaID, which is a fixed-length identifier used in Kademlia networks. The KademliaID is represented as a 20-byte array. Key functions include NewKademliaID, which creates a KademliaID from a hex string, and NewRandomKademliaID, which generates a random ID. The CalcDistance method computes the XOR distance between two IDs, which is crucial for Kademlia's distance-based routing. Other methods include Less and Equals, which are used for comparing IDs, and String, which returns a hex string representation of the ID.

- **CLI:**
  Holds: Kademlia, reader, writer
  Manages: Continuously listen to input from user and reacts accordingly to commands from user such as GET, PUT and EXIT and perform such tasks on the kademlia node it is running on.

- **Main:**
  Manages: Start up of the kademlia node as a "normal" node or a "bootstrap" node accordingly. Connects nodes to the kademlia network. In addition, it starts listening on network, cli and kademlia Action channel as different go routines.

2.2 System Architecture Diagram

See Appendix A for the system diagram and overview of the implementation.

2.3 Frameworks

The frameworks used were the following.

- Go Language
- Go's built in test package; for all tests and to see the test coverage
- GoLand IDE
- Docker

3 Limitations and improvements

3.1 Delitimations of the assignment

The delimitations were decided on to make the task manageable withing the time fame of the course. The system implemented is limited by all of the following.

- Data objects can not be explicitly deleted or modified. In other words, they must be immutable. They can still, however, expire or be lost.
- Data objects can only be requested by their hashes. They have no other names or identifiers.
- Data objects are always UTF-8 strings, which makes it possible to write them into terminals. You are allowed to set an arbitrary string length limit, such as 255 bytes.
- Data objects are not saved to disk, which means that they disappear if you terminate the nodes that hold copies of them.
- No communication is encrypted.
- All network nodes have access to all stored data objects without needing any permissions.

3.2 Limitations and improvements of the system implemented

The following are **limitations** of the system.

- Single Point of Failure (Bootstrap Node): If the bootstrap node fails or becomes unreachable, new nodes can't join the network, limiting network scalability.
- UDP doesn't guarantee message delivery, ordering, or duplicate protection. This could lead to lost messages or incomplete lookups in the DHT.
- There are some hard coded values such as IP address for the bootstrap node.

The following are **improvements** that could be added to the system.

- Prevent the Single Point of Failure by introducing multiple bootstrap nodes to create redundancy. A protocol or mechanism to dynamically select one or switch between available bootstrap nodes could be added.
- A retry mechanism could be implemented for key operations such as FIND_NODE, to detect timeout issues and send the requests again if necessary.
- A config file could have been introduced to add some flexibility and avoid hard coding and easily customize setting such as bucket size and alpha.
- While the CLI is sufficient for the requests required it could have been made a bit more user friendly.

4 Backlog & Source Code

Link to Backlog.

The Backlog contains relevant issues and has been be updated according to the work that has been done. View the different boards for an overview of the work done during each sprint.

Link to Source Code (Repository).

5 Sprint 0

The group members have read the paper on Kademlia together with the lab instructions and gathered a good understanding about the theory behind the algorithm.

All group members have downloaded and installed Docker and Go on their computers. The provided code has been forked to create a new repository for the group. A Github project has been added in order to create a backlog for issues etc.

With the provided code, 50 containerised nodes can be spun up. Using the terminal, the nodes can PING and communicate with each other.

5.1 Plan for Sprint 1

The first step is to setup the Network and implement PING according to the Kademlia specification to make sure nodes can communicate with each other using Kademlia specific code. In order to deploy this, the procedure of how to do this using Docker and GO will be looked into.

Next step is to implement the NODE LOOKUP function as this is a central part of the Kademlia Algorithm. In order to handle concurrency and race conditions, the GO channels and routines will be studied and used.

If time permits for Sprint 1, the storing and finding objects functions will be implemented. It is unclear if the implementation of CLI with the put, get and exit functions should be done before this step. However, this is to be figured out.

Testing for all functionalities will be implemented simultaneously and the procedure on how to create Unit tests in Go will be studied.

6 Sprint 1

6.1 Progress Done

During Sprint 1, the following has been done:

- Spinning Up/Down the network: Some scripts has been written that either spins up all Kademlia nodes as docker containers, stops the nodes or takes them down and deletes the docker image
- Nodes: Assigning a routing-table, network and storage for all Kademlia Nodes + joining the nodes to the network
- Network: Constructing messages, sending them over UDP between nodes etc
- Bootstrap Node: A node with static IP address which all nodes know about during startup
- Kademlia Functions: The Ping, Find Node, Find Value has been implemented. Details may be missed, but overall functionality is done.
- Node Lookup: The iterative procedure of locating a targets k closest nodes has been started.
- User Input: A temporary solution to read user input has been implemented for manual testing of e.g. Ping.
- Unit Testing: Some tests of adding contacts to routing table and making sure the actually closest contacts given a certain target is returned.

6.2 Plan for Sprint 2

A lot of core functionalities are in place such as sending messages such as ping, store and find node. However the overall flow for the functionalities needs work. For example, we need to think about and make sure the exchange of knowledge between nodes communicating works as intended. Another thing that needs some work is that the buckets are updated and handled as intended for example when they are full. So the key focus for Sprint 2 will be to tie things together basically. Currently there are errors and issues with the iterative node lookup. Depending if we manage to solve the problems until the sprint review this will also be continued working on.

Functionality for a node to join the network is not implemented yet and neither is the CLI. Another thing that needs to be done in Sprint 2 is making sure the test coverage is enough. When those things are completed the mandatory objectives M1-M5 should be completed. Last thing is to finalise the report and create a system architecture diagram and description (M6).

If there is time another thing to do is to rethink the network component. There are some refactoring that could make sending and receiving messages a bit simpler than how it is currently working. If there is time we will look at some qualifying objectives also.

7 Sprint 2

Most of the time spent was to tie together the main functionality of the network and get Node Lookup to work. Functionalities like using the PING message to handle full buckets was also implemented.

When the Node Lookup seemed to work and Nodes could correctly be added to the network a strange occurrence was discovered. What happened was that most of the nodes joining the network were initialised via the Bootstrap Node in a correct manner while some of them did not get any information after performing the Node Lookup on itself. After looking through and making sure every error was logged one was found that indeed was not logged. The buffer size when reading/writing to the UDP ports was not big enough and the full string of contacts to be added was not sent as expected.

When this was discovered, there was not too much time left to add qualifying objectives as we had hoped for and the remaining time was spent with some refactoring, clean up of code and writing the last unit tests that was needed to reach the test coverage that was expected.

7.1 Progress Done

During Sprint 2, the following has been done:

- Using ping to handle full buckets
- Solved issues with Node Lookup
- Tied together main functionality
- Debugging
- Refactoring for readability of the code

8 Resources

Implementation details of Kademlia
Create a UDP Client and Server using Go
Net package documentation
Static IP of Docker Container
Using volumes in Docker
Get outbound IP of container in GO
Key Value Pair in GO
String to Data Array Go Startup order of docker services Concurrency in GO

A System Architecture