CPD project 2

Distributed and Partitioned Key-Value Store

Margarida Cosme 201709304
José Pineda 202111200
Tomás Fidalgo 201906743

Index

Distributed and Partitioned Key-Value Store **1**

Problem description **3**

Membership Service **3**

- Store 3
- TCP 3
- UDP (Multicast) 4

Storage Service **5**

- Node initialization 5
- Join Event 5
- Leave Event 5

Problem description

The purpose of this project was to develop a distributed key-value persistent store system, based on Amazon's dynamo, for a large cluster. The system has data-store nodes, and also a test client.

The system stores objects, composed by a text value, and a key, generated using SHA-256 (from the value) on their respective data-store nodes. To ensure persistence, the data items and their keys are stored in persistent memory. The objects can also be consulted by the get method, and deleted.

The nodes that compose the cluster can leave and join, readjusting the storage of the key-value pairs.

Membership Service

Store

The Store is the main class of the project, where the store nodes that compose the cluster, and all their methods are implemented.

TCP

The communication between the TestClient and the Store class is implemented with TCP. The different TCP channels used by the node are handled by different threads of the tcpHandler class.
A client can send the following messages to a node:

- **Join**
  - Format: <node_ap> join
  - Effect: The node calls the join() function and enters the cluster
- **Leave**
  - Format: <node_ap> leave
  - Effect: the node calls the leave() function and leaves the cluster
- **Put**
  - Format: <node_ap> put <file_path>
  - Effect: The node
    - Checks if its in the cluster, if it's not it returns a: "The node you are trying to contact is not in the cluster" message;
    - Calls the getFileValue(path) to read the value from the file;
    - Calls the generateHasCode(value) to get the key;

        - Calls the put(key,value) function, that calls getResponsibleNode(key) function, if it is responsible for that key-value pair, it stores it and returns a “Success” message. Otherwise it returns a message to the client of the type: “The node responsible for that file is <node_id>”
- **Get**
    - Format: <node_ap> get <SHA-256_key>
    - Effect: The node
        - Checks if its in the cluster, if it’s not it returns a: “The node you are trying to contact is not in the cluster” message;
        - Calls the get(key) function, that calls getResponsibleNode(key) function, if it is responsible for that key-value pair, it returns its value in a message to the client. Otherwise it returns a message to the client of the type: “The node responsible for that file is <node_id>”
- **Delete**
    - Format: <node_ap> get <SHA-256_key>
    - Effect: The node
        - Checks if its in the cluster, if it’s not it returns a: “The node you are trying to contact is not in the cluster” message;
        - Calls the delete(key) function, that calls getResponsibleNode(key) function, if it is responsible for that key-value pair, it returns its value in a message to the client. Otherwise it returns a message to the client of the type: “The node responsible for that file is <node_id>”.

UDP (Multicast)

The communication between the different nodes within the cluster is implemented with UDP. The multicast channel used for group communication is handled by the class mcastHandler. A node can send the following multicast messages to the cluster multicast socket:

- **Join**
    - Format: JOIN:<node_id>:<membership_counter>
    - Effect: When a node receives a multicast JOIN message from another node it:
        - Creates a String: “<node_id> <membership_counter> \n” that is essentially a membership_log with only one entry;
        - Calls the updateMembershipLog(log) function that compares the membership log of the node with the log passed as an argument and updates the node’s log;
        - Calls the getFilesName(path) function, that returns an array with all the file names in the node;
        - For each of the key-value pairs in the node (each file), it calls the getResponsibleId(key) function, if the node responsible is no longer itself. it sends a multicast PUT message described below and deletes the key value pair.
        - Calls the getFileValue(path) function with the membership log path. The function returns the file value in a string of the type: “<line>\n<line>\n…”
        - Sends a multicast MEMBERSHIP message described below.
- **Leave**
    - Format: LEAVE:<node_id>:<membership_counter>

    - Effect: When a node receives a multicast LEAVE message from another node it:
        - Creates a String: “<node_id> <membership_counter> \n” that is essentially a membership_log with only one entry;
        - Calls the updateMembershipLog(log) function that compares the membership log of the node with the log passed as an argument and updates the node’s log.
- **Membership Log**
    - Format: MEMBERSHIPLOG:<log>
      Where log is a string with the format: <node_id> <membership_counter>\n
    - Effect: When a node receives a multicast MEMBERSHIP message from another node it calls the updateMembershipLog(log) function that compares the membership log of the node with the log passed as an argument and updates the node’s log.
- **Put**
    - Format: PUT:<responsible_node_id>:<key>:<value>
    - Effect: When a node receives a multicast PUT message from another node it checks if the responsible_node_id is its own id, if it is it calls the put(key,value) function.

Storage Service

Node initialization

A node is initialized with a membership_count of -1, and an empty folder with the node’s hashed id is created in the storage system.

Join Event

A join event is caused by a TCP join message, which calls the join() function.

- The node checks if its membership_count is either 0 or even. If it is its returns an “Error: node is already in cluster” message;
- Otherwise it creates a membership_log.txt file in its directory with an entry for its id and membership count.
- Then it sends a UDP multicast JOIN message, with an effect in the other nodes described above.

Leave Event

A leave event is caused by a TCP join message, which calls the leave() function.

- The node checks if its membership_count is odd. If it is its returns an “Error: node is not in cluster” message;
- Otherwise it updates its own membership log with an odd membership count (using the updateMembershipLog(log) function).

- Calls the getFilesName(path) function, that returns an array with all the file names in the node;
- For each of the key-value pairs in the node (each file), it calls the getResponsibleId(key) function, sends a multicast PUT message described above and deletes the key value pair.
- After deleting all its key-value files, it deletes its membership log file.
- Finally it sends a multicast LEAVE message to the other nodes, as described above.

All the functions were tested independently and have no issues, however, our implementation of the UDP multicast system is flawed and not all the multicast messages are sent successfully. Because of this there are some issues with the key-value store system.