RafBook

Distributed system for managing text files and images

Mihajlo Čumić

1 Introduction

This document describes a distributed system, RafBook, design for managing text files and images that are ASCII encoded. This document gives a detailed overview of the system as well as guidelines for its implementation and usage. Implementation of this system is based on the DHT Chord protocol.

2 Identification

This section explains how entities are distinguished from each other using a custom hashing algorithm.

2.1 Node identification

Every node in the system is identified by the following parameters:

- Universally unique identifier (UUID)
- IPv4 address
- Port number
- Chord identifier

Chord identifier for nodes is its adress and ports hashed value.

2.2 Message identification

- Universally unique identifier (UUID)
- Message type
- Message text
- Sender node address and port
- Receiver node address and port

2.3 File identification

- Universally unique identifier (UUID)
- File name
- File type
- File content

- Chord identifier

2.4 Backup identification

- Owners chord Id
- Location to backup file
- Files name
- File type

3 Communication protocol

This section explains algorithms and procedures that every node in the network must follow to function.

3.1 System configuration

When a node is initializing it reads a local configuration file, which has the following properties:

- Chord size – Number of chord ids that can bi assigned to nodes and files.
- Root - Absolute path for a working directory.
- Bootstrap port – Predefined bootstrap process socket port.
- Nodes port – Node process socket port.
- Low waiting time– Time interval before a node is checked for a failure
- High waiting time– Time interval before a node is declared failed

3.2 Joining a system

Every node that wants to joining the system must be aware of predefined parameters necessary for establishing a connection with the bootstrap server. These parameters are specified in the local configuration file mentioned in the section above. The bootstrap server is a dedicated node used solely for connecting new nodes to the existing nodes in the network and keeping record of all nodes in the system.

When bootstrap server receives a HAIL message from a node that wants to join the system (this is a new nodes first contact with the system), HAIL message response is an integer value, there are two cases:

- returned value is -1, this means that this is the first node that is added to the system
- returned value is random nodes port number, bootstrap randomly selects a node to which the new node will report

If new node gets some nodes port from bootstrap server then it sends NEW_NODE message to that node. If the node is not responsible for new node, then it sends it NEW_NODE message to its next node. When the node which is responsible for the new node adds it to the system it sends back to

the new node WELCOME message. After the new node receives WELCOME message it sends NEW message to bootstrap server and with that it is added to list of active nodes in the system.

3.3 Virtual file system

The purpose of this system is to enable us to work with ACII coded text files and images. This is achieved with the following CLI commands:

- **put [id] [file-path]** – puts a file in to the system. First argument is an integer number from 0 to chord_size, this value will act as a chord_id for this file. Next argument is a relative file path from the root (system supports .txt, .jpg and .png files). If the arguments are valid then the node which got that command adds this file to his values if it is responsible for it and if files id does not collide with some other value then it sends BACKUP message to the next node, if it is not responsible for the new file it sends a PUT message to the next node.
- **get [id] –** gets a file with the passed id. If the file with this id exist on this node it will return that file and put it into the nodes /Downloads folder. If this node is not responsible for this file it sends ASK_GET message to the next node. The node which is responsible for this file sends TELL_GET message to the node that asked for the file. If the file with this id does not exist in the system it simply tells that it does not exist.
- **delete [id] –** deletes a file with passed id from the system and all backups that contain this file. If this node has a file with this id it deletes it and sends REMOVE_FILES_FROM_BACKUPS message to the next node. If it does not contain this file it send ASK_DELETE message to the next node. The node which has this file sends TELL_DELETE message to the node that got the delete command.
- **view-files [address:port] –** lists all the files that are stored on the node which is located on the passed address and listens on the passed port. If the passed arguments represent this node then it just lists its files. If it is not this node then it sends ASK_VIEW_FILES message to the next node for the chordId that id which is obtained by hashing passed port. Then the node which files are requested send back TELL_VIEW_FILES message.
- **data-info** – lists all information of this node, such as stored files, backup file locations, its chordid, all nodes and all successor nodes.
- **pause [milliseconds]** - pauses this node for the passed milliseconds.
- **stop –** stops execution of this node.

4 Fault detection

Node detection is done in two phases using parameters low_waiting_time and high_waiting_time from the configuration file. Every node is monitoring all of its successor nodes if they exist. First stage of monitoring is sending HEARTBEAT_REQUEST messages to all monitored nodes. Then the thread from which monitoring is done waits for low_waiting_time. In the meantime nodes that got HEARTBEAT_REQUEST message respond to it by sending HEARTBEAT_RESPONSE message to the node that sent the request. When the node gets HEARTBEAT_RESPONSE message it updates the the value to true in the list that stores the information if the nodes has responded to the

heartbeat. If all nodes have responded to the heartbeat it does not do anything else and waits for some time to start the next monitoring cycle.

The second stage happens when some nodes have not responded to the heartbeat request and low_waiting_time has passed. Then node gets one of the nodes that have responded and send them RECHECK_NODE message, node that gets this message send ASK_HEALTH_CHECK message to the node that needs to get checked. If the suspicious node gets this message it sends back TELL_HEALTH_CHECK message which will update his value responded value to true and that way the node that monitors will know that it is a healthy node. If the suspicious node does not respond then it stays suspicious. If the monitoring node does not have any healthy node that has responded it will skip the next stages waiting and jump right to removing the nodes part.

After rechecks start the third stage the node wait for high_waiting_time and then check all nodes that it is monitoring and if some still has not responded it sends REMOVE_NODE and LOAD_BACKUPS messages to all healthy nodes and loads backups of that node if they are present on the current node.

At any time nodes that are being monitored by the node can be changes that is why the node keeps the information about the version of his successor table. Every time successor table is updated it increments its version and then heartbeat mechanism will skip the third stage and wait to start the next cycle.

5 Fault tolerance

Fault tolerance is achieved by storing backups of files, each file when it is added to the system is replicated 3 times. Node that stores the file knows the location of the first replica in the case that node on which that backup is being stored exits the system it will make another backup of that file.

Every node can store some other nodes files as backups, backups are files that are written on disc. Backup in the system is an object that stores the path to the backup file and all needed information for reconstructing that file when its loaded in to memory and its current owners chordId. On disc backup file written with the name file_chord_id.files_extensio.

When a node gets removed from a system every healthy node gets a request to load backups of the removed node if the node has them. If the node contains the backups it loads backup file from the disc and reconstructs the file and puts it in the system the same way as explained in the section 3.3.

When a new node is added to the system it sends REMOVE_BACKUPS message which goes around the system and removes all the backups of files that are now stored on this node, because new backups will be made.

When a file is deleted from the system, the node that stores the file sends REMOVE_FILE_FROM_BACKUPS message that will delete all backups that are storing that file.

6 Messages

Following messages are all the messages that the nodes can get and send:

- **HAIL** (3.2) **–** String that contains “HAIL:new_node_port”, this is sent only to bootstrap server

- **NEW** (3.2) **–** String that contains “NEW:new_node_port”, this is sent only to bootstrap server
- **REMOVE** (3.2) – String that contains “REMOVE:nodes_to_remove_port”, this is sent only to bootstrap server
- **NEW_NODE** (3.2) **–** no additional arguments
- **WELCOME** (3.2) **–** Map, key is of type integer and value is File.
- **BACKUP** (3.3) **–** String that contains “n:serialized_file”, n indicates which replica number is this.
- **PUT** (3.3) **–** integer key and String value that is serialized file
- **ASK_GET** (3.3) **–** String value of a key that is requested.
- **TELL_GET** (3.3) **–** integer key value and String of serialized GetResult:
  - UUID
  - integer result_status (1, -1, -2)
  - File file
- **REMOVE_FILES_FROM_BACKUPS** (3.3, 5) **–** String value of a key of a file.
- **ASK_DELETE** (3.3) **–** String value of a chord id of a file.
- **TELL_DELETE** (3.3) **–** Serialized value of DeleteFileResult:
  - UUID
  - integer result status (1, -1, -2)
  - String file name
  - integer port of a node that stored that file
- **ASK_VIEW_FILES** (3.3) **–** String value of a nodes port.
- **TELL_VIEW_FILES** (3.3) - Serialized list of ViewFilesResult:
  - UUID
  - integer key of file
  - String file name
- **HEARTBEAT_REQUEST** (4) **–** No additional arguments
- **HEARTBEAT_RESPONSE** (4) **-** No additional arguments
- **RECHECK_NODE** (4) **–** String value of a nodes port that needs to be rechecked
- **ASK_HEALTH_CHECK** (4) - No additional arguments
- **TELL_HEALTH_CHECK** (4) - No additional arguments
- **REMOVE_NODE** (4) **–** String value of a nodes port that will be removed
- **LOAD_BACKUPS** (4) - String value of a nodes port whose backups will be loaded

- **REMOVE_BACKUPS** (5) – String that contains “nodes_chord_id:serialized_file”