# PAD Project: Distributed File System

Luca Rinaldi

18 March 2016

## Contents



## 1 Introduction

The aim of the project is to create weak consistency distributed file-system by the use of gossiping, consistent hashing and vector clocks.

The communication between node exploit the Java socket mechanism in such the project can be execute in different ways: on a single machine with the use of threads, on a cluster of servers or in virtual containers using *docker*.

The file-system is implemented as a map with a string key and a number or string value with the following operations:

- **add**(key, value), add only if the key is not present.
- **get**(key), get the value of the key if present.
- **update**(key, value), update the key with the new value only if the key already exists.
- **remove**(key), remove the key if present.

# 2 Logical Structure

The file-system is composed by two fundamental parts the front-end, that provide the external access to the file-system through a Restful json API, and the storage system its self where the data are stored and manage. The first part doesn't have any information of the data present on the system, it only know the servers present in the down part.

## 2.1 Communication system

All the internal communication are done with the UDP transport protocol, to avoid the overhead of the TCP and by assuming a good network between nodes.

All the node either the front-end one and the storage one use the gossip protocol to update the list of the servers involved in the file-system. So that each node have two service running on different ports, one for the gossip protocol and the other to receive messages from the other nodes.

When a new request arrive to a front node it is randomly send to storage node from its list and than the node wait 5 seconds for an acknowledgement that the request is correctly served, otherwise it assume that something goes wrong.

## 2.2 Storage protocol

All the storage nodes use consistent hashing to assign a key datum to a given server with the following strategy:

- a server is master for all the data with lower or equal hash value.
- each datum is replicated to a fixed number of next server in the consistent hash circle.

The system use a single master storage protocol without consensus, so the data are write or read without wait for an acknowledgement from the backup's servers.

Each time a new server arrive it immediately became master for the key with a lower hash and a backup server for the keys owned by the previous servers. So after the neighbors discover the new server, they either send the keys that it have to manage or the key that it have to keep for backup.

Within the data it is also added a vector clock to keep trace of the update of the datum. The vector clock is implemented as a map with as key the server id and for value a counter, in this way all the serves that doesn't have a key are consider 0. Then each time a server update a datum as a master it increment the counter with its id inside the object, and foreword this new vector with the datum to his backup server.

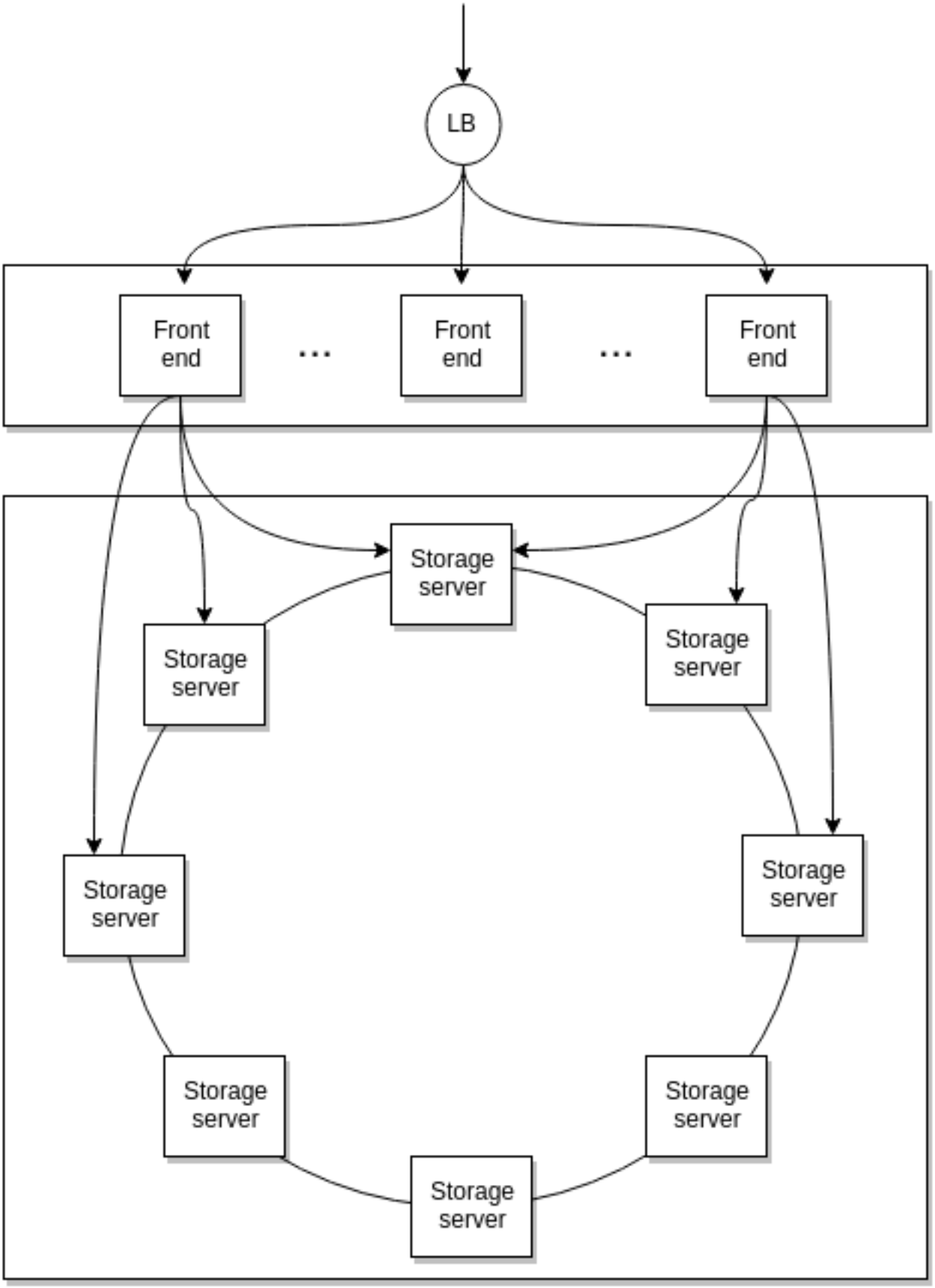


Figure 1: Project logical structure

This vector clock is used every time two version of a value are founded, after some key management. If two unconfrontable version of the data are founded the node server keep the one in which the sum of all the counter is greater, this is a simple heuristic base on the fact that it is more probable that a datum that was updated more times it cam be the newer.

# 3 Project Structure

The system is structured in the following projects:

- **core**, it is a single storage node with all the structure and the essential algorithms to work.
- **api**, it is a single front-end node
- **app**, it implement a distributed file system with a single API server, where each node run on threads.
- **webapp**, this is a nodejs application that graphically show the state of the file-system.

## 3.1 Core

The main class in the core project are:

- **Messages**, and his children class - MessageManage, MessageRequest, MessageResponse, MessageStatus - they all represent a json message exchanged between the file-system nodes.
- **ConsistentHash**, the implementation of a Consistent hash owned by each storage nodes.
- **Data**, it represent a generic datum saved inside the system.
- **Node**, a generic server node of the system either a front-end or a backed server.
- **StorageNode**, it is a storage node object.
- **GossipResurce**, it is the resources use by the front-end to exploit the gossip protocol.
- **PersistentStorage**, a wrapper for the MapDB library used to persistently store data on the service node
- **VectorClock**, the implementation of the vector clock included in each data

### 3.1.1 Node class

The node class represent a server of the file-system with the id, the ip and the port of either the gossip server and the management service. It also expose a `send(Message msg)` method to send to it a new message with the `DatragramSocket`.

This class is extended by the `StorageNode` and the `GossipResurce` which implement a storage node and a front-end node.

The `StorageNode` can be instantiated with the constructor `NodeService(String id, String ipAddress, int port, List<GossipMember> gossipMembers)` that initialize all the needed structure such as:

- `PersistentStorage` class
- `DatagramSocketServer` with use to receive message from the other node
- `ConsistentHash` class,
- `GossipService` to maintain the consistent hash structure

It has a thread that continually check for new message from the other nodes, and for each one it distinguish two general case: if it is a request from another node (either a front-end node or a request pass from another StorageNode) or if it is a `MessageManage` that indicate some manage operation send send by the master of a key.


Figure 2: screen-shoot of the MonitorWebApp

### 3.2 Api

The Restful API, implemented with the Spring web framework, exposes the two end point `/api` and `/status`. The first is the public entry-point to operate with the file-system. The second is a monitoring tool that get a snapshot of all the node in the file-system with their data structures.

The following operation can be used on the `/api` resources:

- get a key, `method: GET, parameter: key`
- add a key, `method: POST, body: {"key": "..", "value": ".."}`
- update a key, `method: PUT, body: {"key": "..", "value": ".."}`
- delete a key, `method: DEL, parameter: key`

Each of the previous operations return a json object with the `status` field that can be either `ok` or `error` and the `data` field with optional return data.

## 4 How to use

It's possible to use the distributed file-system in the thread version for a single machine, the multi server for a cluster of machine or using the docker container either in a single machine or in cluster.

The simplest way to use the file-system is to download the last release and run it with the only requirement of Java8. In this way the application can be used either in the thread version with the `app-<version>.jar` or in the cluster version with the storage node `core-<version>.jar` and the front node `api-<version>.jar`. In the release is also possible to find the the MonitorWebApp for Linux, MacOS or Windows.

#### 4.0.1 Requirements

- **java8**
- nodejs/npm (optional only for the MonitorWebApp)
- docker>=10 (optional only for the docker version of the file-system)

### 4.1 Thread version

It can be build with:

```
./gradlew app:build
```

and run with:

```
java -jar app-<version>.jar
```

Optional parameters:

- `-N <number>` number of servers to start
- `-n <number>` number of seeds servers
- `-gport <number>` the port number used by the gossip protocol
- `-mport <number>` the port number of the management server

example:

```
java -jar -N 10 -n 2 -gport 3000 -mport 2000"
```

### 4.2 Single server

It can be build with:

```
./gradlew core:build api:build
```

To start a storage node run:

```
./java -jar core-<version>.jar
```

To start a front server run:

```
./java -jar api-<version>.jar
```

optional parameters:

- `-id <string>` the port of the server
- `-ip <string>` the ip of the server
- `-p <number>` the port of the server (two successive port are used)
- `-m <id>:<ip>:<port>` a seed server to start the gossip protocol (use this parameters for each seed server)
- `-h <id>:<ip>:<port>` all the server configuration as a single string

example:

```
./java -jar core-<version>.jar -h server1:192.0.0.5:2000 -m server2:192.0.0.2 -m server3:192.0.0.3
```

### 4.3 Docker version

to build the docker image of the front node and the storage node run

```
./gradlew core:build core:docker api:build api:docker
```

Now is possible to execute a demo by run the following perl script:

```
perl start-docker.pl <number of storage node :default 5>
```

To manually run a file-system node you have to create a new docker network with the command:

```
docker network create --subnet=172.18.0.1/16 fs-net
```

then to start a two node file-system run

```
docker run -d --net fs-net --ip 172.18.0.1 pad-fs/core:0.1 -h server1:172.18.0.1:2000
docker run -d --net fs-net --ip 172.18.0.2 pad-fs/core:0.1 -h server2:172.18.0.2:2000  -m server1:172.18
```

and a front end node with

```
docker run -d -p 8080:8080 --net fs-net --ip 172.18.0.20 pad-fs/api:0.1 -h rest:172.18.0.20:2000 -m serv
```

### 4.4 MonitorWebApp

The webapp can be used with one of the release version for the different OS or run with the nodejs interpreter with the following command:

```
./gradlew webapp:run
```

## 5 References

### 5.1 Java libraries:

- edwardcapriolo/gossip (https://github.com/edwardcapriolo/gossip)
- MapDB (http://www.mapdb.org/)
- FasterXML/jackson (https://github.com/FasterXML/jackson)
- JUnit (http://junit.org/)
- Spring (https://spring.io/)
- JCommander(http://jcommander.org/)

### 5.2 nodejs and javascript libraries:

- Angularjs (https://angularjs.org/)
- Bootstrap (http://getbootstrap.com/)
- jquery (https://jquery.com/)
- nwjs (http://nwjs.io/)

### 5.3 build tools:

- Gradle (https://gradle.org/)
- gradle-docker (https://github.com/Transmode/gradle-docker)
- npm (https://www.npmjs.com/)
- nw-builder (https://github.com/nwjs/nw-builder)

### 5.4 docker images:

- java (https://hub.docker.com/_/java/)