PAD-FS

Leonardo Gazzarri

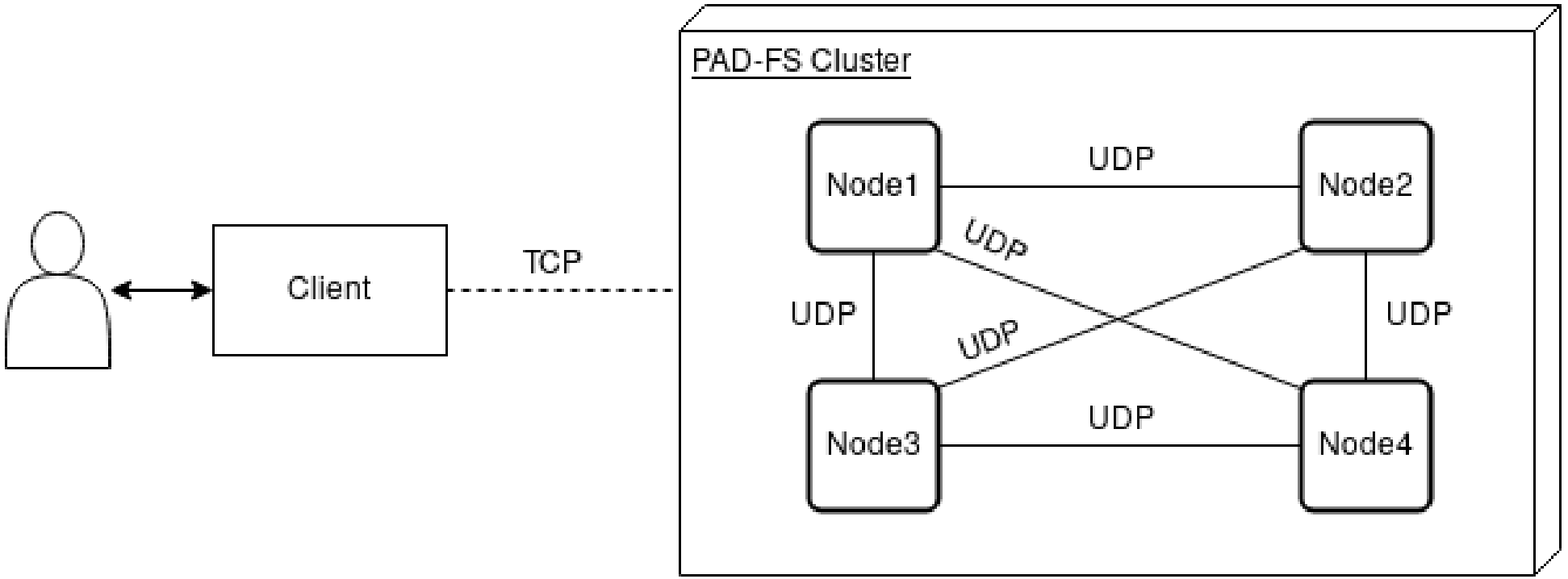
Overview

- A distributed key-value storage system
- Simple API: put, get, list, remove
- CLI interface
- Flat
- Source code available at: https://github.com/Murray1991/PAD-FS

Design choices

- Data Partitioning using consistent hashing technique with virtual nodes
- Passive replication architecture
- Anti-entropy protocol for replica synchronization
- Data Versioning with vector clocks facilitate consistency
- Gossip protocol for membership and failure detections

Architecture

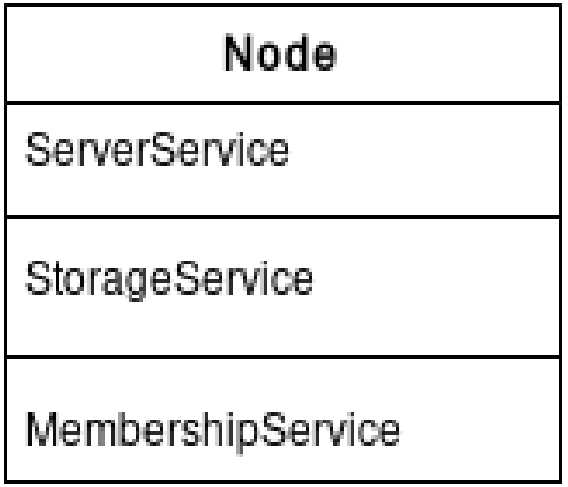


Client

- Offers a CLI interface to the user for *put*, *get*, *remove*, *list* operations
- It could not have the complete view of the storage system
- Communicates via TCP with a PAD-FS node according to a classic client – server paradigm

Node

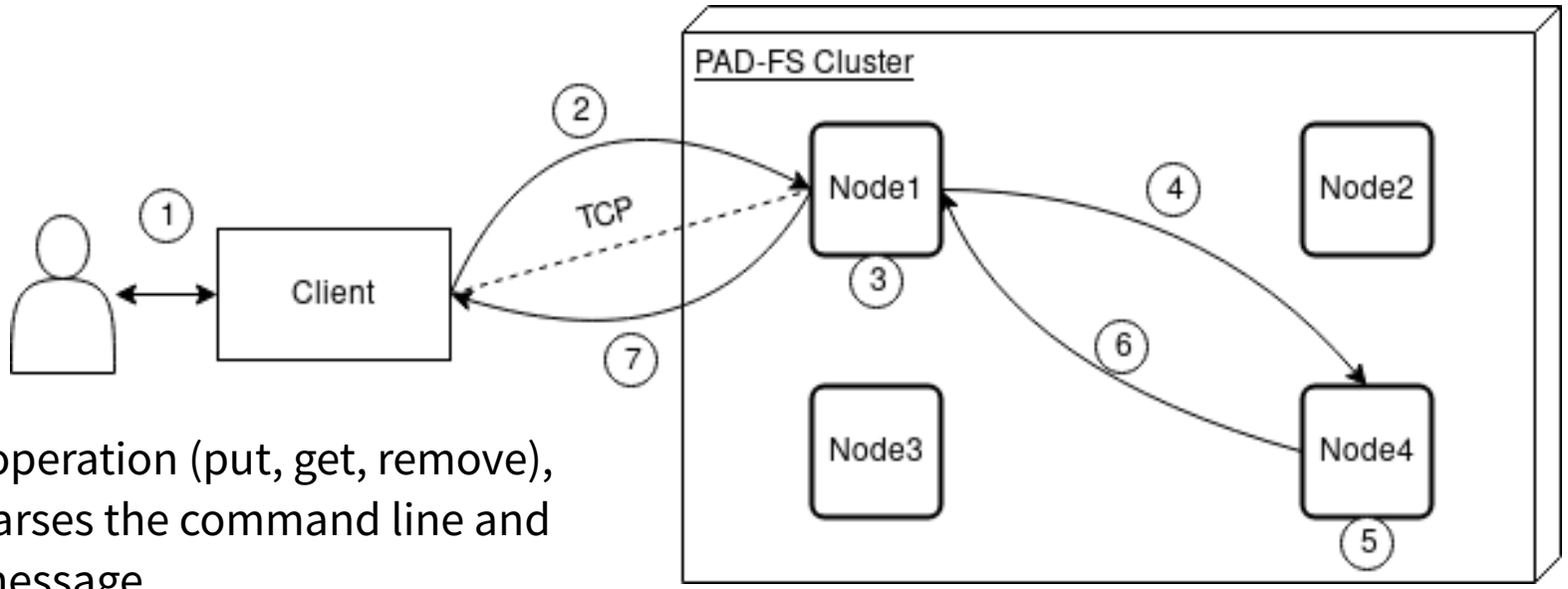
- **The core of the distributed storage system**
- **Composed by several "services"**
  - *ServerService*: communicates with the client via TCP
  - *StorageService*: performs local updates, routing of requests to the primary node and implements anti-entropy protocol for replica synchronization
  - *MembershipService*: implements gossip protocol for membership/failure detection and offers key-node mapping through consistent hashing technique
- **Node-to-node communications use UDP**



The operations

- **The system offers to the client four basic operations**
  - *put(k,v)*: update the system storing the value *v* for the key *k*
  - *get(k)*: get the value associated to key *k*, if present
  - *remove(k)*: remove from the system key *k* and its associated value
  - *list(addr)*: get the list of all the keys for which exist a value in the node with address addr
- **For each key *k* exists only one server (called primary) that processes client requests (passive replication strategy)**

Processing a client request



1) Client asks for an operation (put, get, remove), the client program parses the command line and generates a proper message

2) The client program open a TCP connection with Node1 (chosen at random) and sends the message

3) Node1 receives the message and determines who is the primary node that should process it

4) Node1 routes the message to the primary node (Node4) and awaits for its response

5) Node4 processes the request eventually updating its local database

6) Node4 sends back the response of its processing to Node1

7) Node1 forwards the received message to the client

Processing a client request

- Each time a node receives a client request it spawns a new thread
- When a node update its local store it stores also a copy of the global vector clock indicating the arrival time of the request
- An entry is updated in function of the already present associated vector clock
- Remove is handled like a write in the local store, but before updating the local store the same request is multicasted among all the nodes
- Versioning is important in order to facilitate consistency among replicas and

Processing a client request - Questions

- What happens if the client message is lost in between the proxy and the master?
- What happens if the response message from the master does not arrive in time?
- What happens if the master fails after immediately after an update?

Replication protocol

- A node X periodically picks another alive Y
- X sends to Y all the keys that are present in its local store via one or more push messages. To each key is associated also the vector clock stored with its value
- For each received push message, Y checks via vector clock comparison what keys are not locally updated and sends them back to X via one pull message
- For each received pull message, X sends back to Y one or more reply messages (containing the keys and all the associated versioned values)
- For each key in a received reply message, Y updates / merges its local store entry with the key associated versioned values

Replication protocol

- The view of alive neighbors is given by the running gossip protocol (MembershipService)
- Implemented by StorageManager and ReplicaManager threads
- ReplicaManager is a running thread that periodically picks an alive neighbor and sends to it a sequence of push messages. .
- StorageManager is a running thread that receive messages at the storage layer (client-specific messages and replication-specific messages) and for each kind of message it spawns the proper handler

Possible issues and limitations

- **The client can't send values of any dimension**
- **Each node periodically sends out all the keys in its local store via push messages**
  - Faster convergence but possible scalability problems?
- **In client message processing, inconsistencies could arise if the network drops (or delay too much) packets**
- **The system could be slow in reacting to failures**