BitTorrent Tracker: deliverable 1
Group 01

Irene Díez Jesus Sesma

1 Architectural design

Figure 1 shows the architectural design of the BitTorrent Tracker. All swarm member receive UDP multicast requests from the peers; however, just the master answers. The communication between the peers is done via UDP, and the tracker's instances contemplate UDP/JMS communication among them.

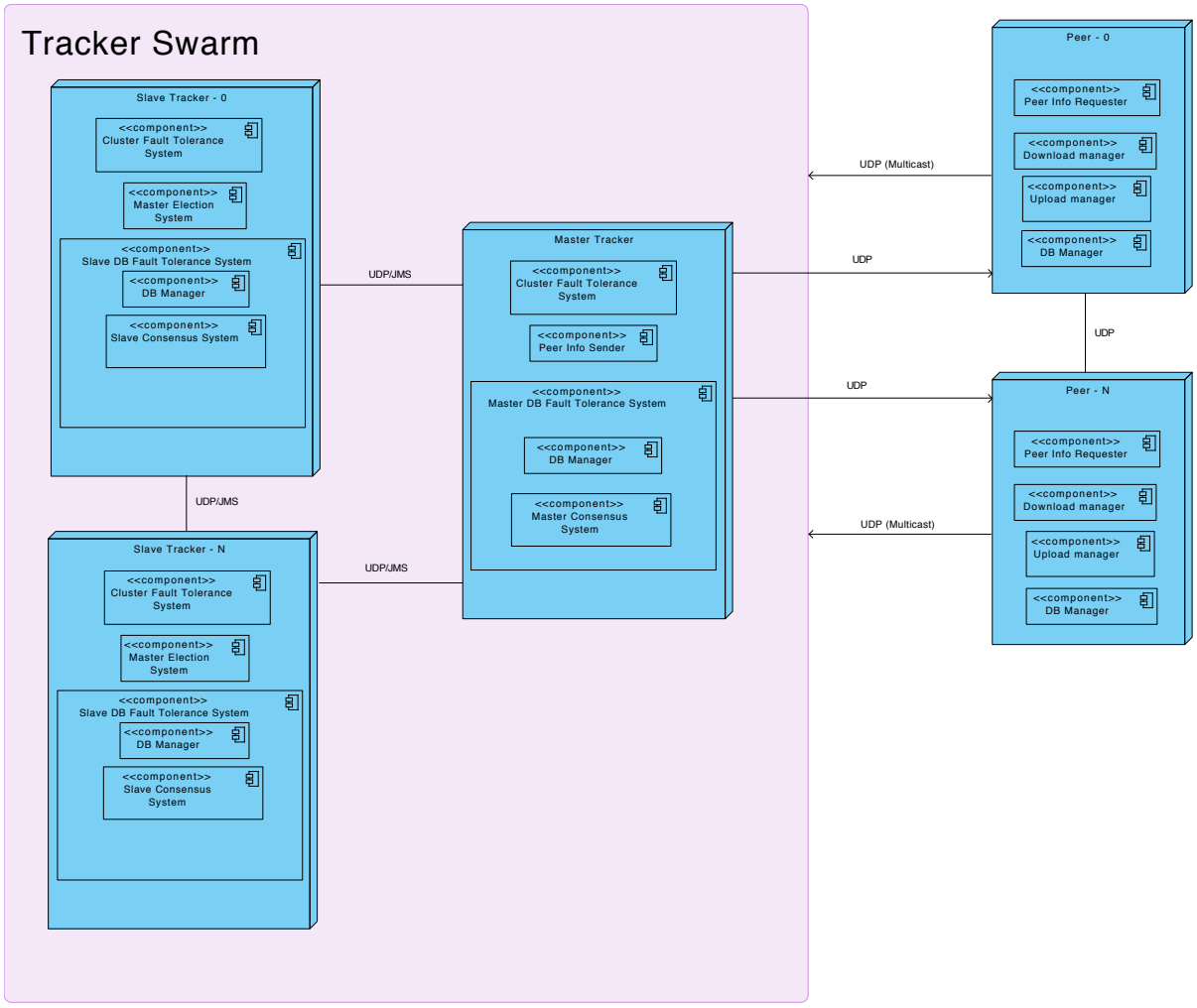

Figure 1: Architectural design of the Tracker.

2 Functionality

The tracker's functionality is summarised in table 1.

The following list describes more thoroughly the functions previously shown in table 1 that each component is responsible for:

Identifier	Entity	Description
Peer Info Sender	Master tracker (MT)	Sends information about the peers and the content they have available.
Master DB Fault Tolerance System (M-DFTS)	MT	Composed of the *DB Manager* and *Master Consensus System*, ensures that the DB is replicated among all the swarm members.
Master Consensus System	MT	This component manages the DB replication.
Master Election System	Tracker slaves (TS)	Chooses a new master among the slaves when the previous one has fallen.
Slave DB Fault Tolerance System (S-DFTS)	TS	Composed of a *DB Manager* and a *Slave Consensus System*, listens to the M-DFTS' orders to ensure the DB replication in the slave.
Slave Consensus System	TS	Ensures the DB replication of the slave.
Peer Info Requester	Peers (P)	Requests information about the peers and the available contents.
Download Manager	P	This component handles the downloads in the clients.
Upload Manager	P	This component handles the uploads in the clients.
Cluster Fault Tolerance System	MT and TS	In charge of sending keep-alive messages among all the members of the swarm.
DB Manager	MT, TS and P	Interface with the DB system.

Table 1: Summary of the functionality implemented in each entity.

- Master tracker
  - *Cluster Fault Tolerance System*: this component is implemented in all the tracker's instances, and it is in charge of knowing the state of all the instances. This component sends and receives Keepalive (KA) messages from all the instances of the cluster. When the master dies sends a notification to the *Master Election System* to start the new master election process.
  - *Peer Info Sender*: answers to a client's information request sending information about the available peers with a specific content.
  - *Master DB Fault Tolerance System*
    - *DB Manager*: handles the Master-Slave database schema described at Section 3, see figure 2.
    - *Master Consensus System*: it has two main functions, (i) when a DB transaction needs to be propagated among all the instances, this component is in charge of coordinating such event by waiting for all the instances to be ready, and transmitting the order; (ii) when a new slave is created it sends the necessary information to keep its DB up to date.
- Tracker slaves
  - *Cluster Fault Tolerance System*: refer to the Master Tracker's description of this item.
  - *Master Election System*: it is notified by the *Cluster Fault Tolerance System* about a failure in the Master. This component selects a new master among the available slaves. When activated, and while the Master election process lasts, the tracker will not listen to the clients requests.
  - *Slave DB Fault Tolerance System*
    - *DB Manager*: refer to the Master Tracker's description of this item.
    - *Slave Consensus System*: this component has two functions; on the one hand (i) when the *Master Consensus System* requests an operation to be propagated, checks the slave's status and prepares it to do such operation; when the slave is ready, notifies the master, and finally, when the master orders to commit, complies. On the other hand, (ii) when the slave is a brand-new instance, it requests the latest DB information to the master and waits for its instructions to commit.
- Peers
  - *Peer Info Requester*: requests information to the tracker about the available peers with some content.
  - *Download Manager*: manages the download process.
  - *Upload Manager*: manages the upload process.
  - *DB Manager*: handles the Peer database schema described at Section 3, see figure 3.

3 Data schema

Our system contemplates two different database schemas, on the one hand the tracker's master and slaves will implement the schema shown in figure 2. This schema has two tables: (i) `PEER-INFO` where information regarding the peers' host and ports is stored; and (ii) `CONTENTS`, where the tracker will store which peers have available some specific content. Both tables' fields are self-descriptive.

```
PEER-INFO (id:INTEGER, host:VARCHAR(255), port:INTEGER)
CONTENTS (sha1:STRING(40), peer_id:INTEGER)
```

Figure 2: DB schema for the tracker's master and slaves.

On the other hand, the peers will need to remember the progress they have made during a download; thereby, they will store which chucks they have downloaded so far for a specific file. It must be underlined that our system will always transfer chunks of the same size, with the exception of the last chunk, or when the content's size is inferior to our default chunk size.

This characteristic is implemented using the schema shown in figure 3, with the table `CHUNK`; its fields are self-descriptive.

```
CHUNK (sha1:STRING(40), offset:INTEGER)
```

Figure 3: DB schema for the tracker's peers.

3.1 DB technology

We have decided to use SQLite [1] as our storage technology, mainly because we have previous experience with it; but more importantly, because this is a didactic project without the high availability and performance needs that a professional project's database requires.

Regarding the use of a classic SQL database over the NoSQL paradigm, since this project implies the synchronisation of the DB over the trackers, we think that the data should be as normalised as possible and complying to the third normal form (3NF); thereby, we discard NoSQL databases.

4 Interaction model design

- Tracker-Tracker
  1. Keepalive (KA) messages:
  2. Master election (ME) messages:
  3. Database synchronisation (DS) messages:
- Tracker-Peer
  - Connection establishment: a peer requests a connection to the tracker to execute a specific action.
    1. Peer requests the connection with the following packet (128 bits):
       `[connection_id, action, transaction_id]`
       Where:
       * `connection_id` (64 bits): initialised to `0xFEELDEADBAADBEEF`.
       * `action` (32 bits): 0, for connection request.
       * `transaction_id` (32 bits): random number generated by the peer.
    2. Master tracker answers with this packet (128 bits):
       `[action, transaction_id, (connection_id|error_string)]`
       Where, if there aren't errors:
       * `action` (32 bits): 0, for connection request.
       * `transaction_id` (32 bits) the `transaction_id` previously sent by the peer.
       * `connection_id` (64 bits): the `connection_id` generated by the tracker, which will be used by the peer in future requests. The peer can use the same `connection_id` for one minute, the tracker will accept the `connection_id` for a specific peer for two minutes.

       If there are errors:
       * `action` (32 bits): 3, for error.
       * `transaction_id` (32 bits) the `transaction_id` previously sent by the peer.
       * `error_string` (64 bits): string describing the error.
  - Peer announce: a peer requests info about a torrent.
    1. Peer sends the following packet (592 bits):
       `[connection_id, action, transaction_id, info_hash, peer_id, event, ip, key, num_want, port]`
       Where:
       * `connection_id` (64 bits): id given by the tracker after successfully establishing a connection.
       * `action` (32 bits): 1, for announce.
       * `transaction_id` (32 bits): random number generated by the peer.
       * `info_hash` (160 bits): SHA-1 of the announcing torrent.
       * `peer_id` (160 bits): peer's id.

    * `event` (32 bits): status; 0 none, 1 completed, 2 started, 3 stopped.
    * `ip` (32 bits): peer's IP address.
    * `key` (32 bits): unique key generated by the peer.
    * `num_want` (32 bits): maximum number of peers wanted in the reply.
    * `port` (16 bits): the port the peer is listening to.
  2. Master tracker replies with one of these packets:
     Normal response (128 bits (min) to (128 + (48 * 6)) (max) bits):
     `[action, transaction_id, interval, num_ret, [ip, port] ...]`
     Where:
    * `action` (32 bits): 1, for announce.
    * `transaction_id` (32 bits): the `transaction_id` previouslyu sent by the peer.
    * `interval` (32 bits): minimum number of seconds that the peer must wait before reannouncing itself.
    * `num_ret` (32 bits): number of returned `(ip, port)` peer info tuples.
    * `ip` (32 bits): ip of a peer.
    * `port` (16 bits): listening port of a peer.

     If there are errors (128 bits):
     `[action, transaction_id, error_string]`
    * `action` (32 bits): 3, for error.
    * `transaction_id` (32 bits) the `transaction_id` previously sent by the peer.
    * `error_string` (64 bits): string describing the error.

- Peer scraping: a peer sends information about a torrent file.
  1. Peers sends info:
  2. Master tracker replies:

5 Failure model design

6 Graphical interface

References

[1] SQLite. `https://www.sqlite.org/`.