ECE419 Lab 4

Zeeshan Qureshi Jaideep Bajwa

11 April 2013

Requirements

- Ant (build sytem)
- Ivy (dependency management)
- Guava (event bus)
- Gson (json serialize/de-serialize)
- ZeroMQ (Distributed Message Queue)
- ZooKeeper (Distributed Coordination Service)

Usage

Install dependencies and build project:

```
$ ant
```

Run Client:

```
$ ./client.sh {zKhost} {zKport}
```

Run JobTracker:

```
$ ./jobtracker.sh {zKhost} {zKport} {Tracker-Port} {Tracker-Id}
```

Run Worker:

```
$ /worker.sh {zKhost} {zKport} {Worker-Id}
```

Run FileServer:

```
$ ./fileserver.sh {zKhost} {zKport} {Server-Id} {File-name} {Server-Port}
```

Design Decisions

Our design for this assignment contains 4 components Client, Job Tracker, Worker and File Server (each of these components work together and are linked by ZooKeeper)

- Job Tracker, File Server and worker upon start connects with the zookeeper. JobTracker and file server store their IP address and port they are listinig at in zookeeper.
- upon start job tracker, file server and worker create a znode under /tracker, /fileserver, /worker respectively.
- job tracker and file server perform leader ship election and become primary or backup accordingly
- all workers are alike (group membership with zookeeper)
- each client maintains a connection with the Job Tracker.
- each worker maintains a connection with the file Server.
- we rely on watches to assign task and to fetch results.

We have decided to keep JobTracker stateless and all the data regarding tasks is stored in ZooKeeper.

Task Management

Job submission

- client submits the task to the JobTracker with the hash, for eg "job 421493fa48fc8df84d1f5f3478cf247a"
- multiple clients can submit task to the JobTracker.

Job Processing

- upon receiving a task, job tracker creates a znode under /jobs with name equal to the hash given by the client. Job tracker gets the current workers by getting the children of /worker, knowing the number of partitions of the file located in the fileserver the job tracker computes what partition id each worker should on. It serializes all the information (json) and store with the data associated with each job.
- workers upon start sets a watch at the children of path /jobs. When job tracker creates a znode under /jobs, worker's watch is triggered.
- worker gets the children of path /jobs and work on each job. It knows which partitions to work by parsing the data associated with each job.
- worker sends the partition id to the fileserver and fileserver responds with the corresponding data chunk.

- worker computes the hashes and also store in a local data sturcture (cache for subsequent jobs)
- if the worker finds the "password", it deletes the znode /jobs/ and creates a znode under /results with data as the found password.
- if the worker doesn't find the "password" it removes its name from the worker list and write it back to the znode data. In case the current worker is the last one to remove its name then it also deletes the znode /jobs/ and create znode under /results with data as null.

Return results

- client submits a request to fetch the result of a task. For eg. "status 421493fa48fc8df84d1f5f3478cf247a"
- job tracker upon receiving the request gets the data associated with znode /results/ and return it to the client. It data equal to null then password doesn't exists.

Handling failure scenario

Each failure scenario is discussed in the following section.

Primary JobTracker failure

- all the task related information is stored in zookeeper, hence no replication is required.
- clients and backup tracker sets a watch on primary tracker.
- upon failure backup tracker watch is triggered and it becomes primary by setting its IP, Port onto the data of znode /tracker
- clients resets the connection with the new tracker by getting the data of znode /tracker
- jobtracker doesn't interact directly with the workers (via zookeeper) therefore nothing needs to be done.

Primary FileServer failure

- similar approach is followed as in the previous failure scenario. Each worker and backup file server sets a watch on primary file server
- upon failure backup fileserver watch is triggered and it becomes primary by setting its IP, Port onto the data of znode /fileserver
- each worker resets the connection with the new fileserver by getting the data of znode /fileserver

Dynamic Worker addition/removal

- when a worker is added the job tracker accounts for it and assigns it work on subsequent job request. (everytime the tracker assigns task it gets the latest number of children of znode /worker and distribute work to each of the workers)
- when worker dies, watch is triggered on the tracker and the tracker fetches all the tasks that were assigned to that worker. Tracker assigns the task to a worker at random and updates the worker task list and writes it back on the data of the znode /jobs/
- each worker also periodically checks if its task is updated, if so it works on the new partitions assigned to it.