ZooKeeper

Jiaya Carl He 1000646946

He Zhang 1000347546

Task Management

The major design is to include stateless JobTrackers and store all the job state in Zookeeper. The design of the znode architecture with the zookeeper will be discussed in the following.

Upon the initial setting up of a primary Job Tracker, 4 permanent znodes called “**Jobs**”, “**Workers**”, “**TaskWaingtingQueue**”, “**TaskProcessQueue**” will be created.The znode “**Jobs**” will include all the jobs information received in the future (e.g the path of first job will be [Jobs/Job1]); similarly, the znode “**Worker**” include all information of the worker. Under each job(e.g Job1), two znode (“**Success**” and “**Failure**”) will also be created.

For a job tracker, upon receiving a Task, a job Tracker will create a znode with labeled a sequence number under the znode “Jobs”. In terms of Task partition, the job tracker will get the number of worker by calling **getchildren()** of znode, “Workers”. Then it can give the task with an interval. (e.g 1/100 - 5/100) With this interval, the worker are able to identify to fetch which corresponding portion of the dictionary.

In terms of communication between job tracker and worker, two queue (**Task waiting queue** and **Task processing queue**) inside the zookeeper will be used. Once the primary JobTracker receives a job, it will decompose it into many tasks and put these tasks into the **Task waiting queue.** Each time All the workers will be set with a watch on the **Task waiting queue** and workers will be notified if there is a change on the waiting queue. Once workers are notified, a worker will fetch a task from the **Task waiting queue** and label its tasks with its own name and timestamp; then, they will put the task with the label into **Task processing queue.** After that, the worker will fetch the dictionary partition associate with the Task from Primary File Server and perform hashing. Once a worker finish its task, then it will remove the corresponding tasks from **Task processing queue.** If a worker didn’t find the match, then it will put the the tasks to znode “**Failure”** under the specific znode job (e.g Job1) associated with the task. On the contrary, it will put the task to znode **“Success”** if it find the match.

Once receiving a message for checking status from clients, the job tracker will check whether some of jobs have been completed. The criteria to check whether the job is complete is to see if the total number of tasks under **“Failure”** and “**Success**” match the number of tasks assigned originally. In this case, the job tracker will remove the corresponding job node and respond to clients.

Failures

1)Primary JobTracker Failure
The JobTracker is stateless and it stores most of the information in the zookeeper, such as the remaining tasks not posted yet to the **Task waiting queue**. Thus the failure of JobTracker is

trivial except for one thing: it need to watch for every single worker in case the worker fails. Once the primary JobTracker fails, the backup JobTracker will reset the watches on those workers so that if the worker fails the backup job tracker can detect it and then process the failure of worker by reposting the failed job to the **Task waiting queue**.In addition, the backup Jobtracker will detect the failure of primary JobTracker and then connect to zookeeper. It will check the information about how many tasks left to be assigned; then, it will continue post those remaining tasks into the **Task waiting queue.** If the jobtracker failed before it updates the information of remaining tasks to be posted to the zookeeper. It will post some duplicated tasks to the **Task waiting queue**, but this is not a problem since the worker will check the duplicated task when it finish its task. As a result, a small duplicated task won't cause any problems.

2)Primary File Server Failure
The failure of primary file server will cause all of the workers to reconnect to the backup file server once the backup file server connect to the zookeeper. Therefore we need to let every worker watch for the file server znode. If the file server fails, every worker will detect the failure and they will set the watch for the file server again to wait for the new file server to connect and get it's ip address. After they have the new ip address they can reconnect to the new file server and start their request for dictionary partition. If the file server fails during the file transfer process, each worker need to record the failed partition id and then resend the request after the new file server connection. All of the waiting requests need to be paused during the downtime of the file server.

3)Dynamic Worker addition/removal
When a worker took the task and failed before finishing the task. Our design allows incomplete tasks to be reassigned to another worker by using the **Task Processing Queue.** A primary Job Tracker would set a watch on each worker; if a worker fail, the Job Tracker will be notified. Once the Job Tracker know which worker fail, it will search through **Task Processing Queue** and remove all the tasks taken by the worker who fails; then, those tasks will be redistributed back to **Task waiting queue** and the other alive workers will fetch them. We let worker to remove the task from the **Task waiting queue** only after it added to the **Task Processing Queue** to avoid the racing condition that when a worker fetch the task from **Task waiting queue** but it crashed before it posts the task to the **Task Processing Queue**. Therefore if it crashed after it get the task but before it post to the **Task Processing Queue** the task will still be in the **Task waiting queue** and thus another worker can fetch it now because it detect that task is in the **Task waiting queue** but not in the **Task Processing Queue**. The Jobtracker will detect the failure but it will find out that no task associated with that failed worker is in the **Task Processing Queue** so it will do nothing.

4) Combination of failure
<u>FileServer and Worker</u>
The combined failures of file server and worker will be ok since the file server and worker are not closely related with watches or other connection. The worker will establish new socket

connection and fetch the file from file server each time it need to get some files. So in the case when they both fails, they will not know until the worker reconnects to the zookeeper and try to get the file server's IP address but it will found either the file server not connect yet or it is already reconnected. In the first situation it will simply wait until file server reconnect and second case is not a problem.

<u>JobTracker and Worker</u>

a. (JobTracker fail ⇒ Worker Fail while backup JobTracker is on the process of transition)
When the backup Jobtracker become primary, it will also check the Task processing queue because the watch of worker is not be triggered. Therefore it is necessary to check the **TaskProcessing queue** to see if there are some tasks taken by any failure workers. If there are some, it means we need to reassign the task by removing the task from **TaskProcessing Queue** and putting it to **Task Waiting Queue**.

b. (Worker fail ⇒ JobTracker Fail)
The only problem here is that JobTracker may not finish reassigning tasks by the time it crashes. However, the mechanism provided part a) is sufficient to cover this case because the the backup Jobtracker will be able to reassign tasks.