ECE419 Lab4 Design Document

Name: Hsiuwei Tsao, #998072981
Name: Yichong Zhang, #1000285693

Design Choices & Task Management

We implemented stateless JobTrackers and stored all job states in ZooKeeper. The ClientDriver submits jobs to the primary JobTracker, and JobTracker populates ZooKeeper with jobs under the "root" node: **root/job.** The workers get notified when there are new jobs adding on the ZooKeeper. The worker will find the job that is not being processed; then it will create an answer node (ephemeral type) under the corresponding job node: **root/job/answer**, and set the data as "In progress". Workers connect to the primary FileServer, and each one will receive the whole dictionary file. While the worker is receiving the dictionary file, it computes the hash for each word, and put the hashed word (key)/plain word (value) pair on the local HashMap. The worker checks whether the HashMap contains the job (key), and if it does, it retrieves the answer (value). The worker calls setData on the corresponding answer node. Depending on the result of the check, the answer node contains either "Password found: answer", or "Failed: password not found".
When ClientDriver queries "status job", JobTracker checks the answer node under the corresponding job node, and returns either "In progress", "Password found: answer", or "Failed: password not found". If ClientDriver queries a status which it never submits to the JobTracker, the JobTracker will return "Job not found". This is done by calling zk.exists(root/job). If the return Stat is null, JobTracker will know that this job is never submitted before. The name of the job node is the password hash submitted by the ClientDriver. Therefore if the ClientDriver submits a job called "apple", the JobTracker will create root/apple, and one worker will in turn create root/apple/answer. In summary, our design has successfully implemented all the objectives outlined in the lab 4 specification. It is tested manually by submitting various kinds of jobs/queries. All the functionalities are working, and the design is fault tolerance.

Fault Tolerance (Failure Scenarios)

a) Primary JobTracker failure: the primary JobTracker creates an ephemeral JobTracker node, and if it crashes, the backup will make itself as the primary and creates an ephemeral JobTracker node. The ClientDriver has a watch on the JobTracker node, so it will get notified when the primary JobTracker

crashes. It will reconnect to the new primary Jobtracker, and resume what it was doing previously.

b) Primary FileServer failure: this is handled in the same way as above. The primary FileServer creates an ephemeral FileServer node, and if it crashes, the backup will make itself as the primary and creates an ephemeral FileServer node. The worker has a watch on the FileServer node, so it will get notified when the primary FileServer crashes. It will reconnect to the new primary Fileserver, and resume what it was doing previously.

Scaling (Dynamic Worker addition/removal)

Our framework can make progress even if N-1 Workers fail. As long as there is one worker present, the job can be processed, and the answer will be eventually written on the answer node. When a particular worker fails, its job will be reassigned. This is because the answer node under that particular job node is ephemeral. Other workers will notice the answer node is gone, and one worker will pick up that job, create a new answer node, and start processing it.
Whenever a worker is created, it will create a ephemeral worker node under the persistent workerRoot node. The JobTracker can check the number of children under workerRoot, and if all workers fail, the JobTracker will return “Failed to complete job” to the ClientDriver.

Concurrent job executions

Our design supports concurrent job execution because JobTracker can accept multiple clients' requests concurrently, and the jobs will be stored on the ZooKeeper. Workers execute in parallel, and the results will be written back on the answer nodes. Each client can check status concurrently, and will receive the answer back.