In this lab, we provide some scripts to help you run the code and evaluate your implementation. The folder of these scripts also comes with a structure that can be used as a starting point of your implementation. The structure of the folder is listed below:

- lab 4
    - README
    - src
        - Makefile
        - ClientDriver.java
        - Worker.java
        - JobTracker.java
        - FileServer.java
        - lib
            - zookeeper-3.3.2.jar
            - log4j-1.2.15.jar
        - any other files …
    - script
        - start_jobtracker.sh
        - start_fileserver.sh
        - start_worker.sh
        - submit_job.sh
        - check_job_status.sh

Notice that all the four java files listed above are not included in the folder, as it is your job to implement them. By running the Makefile, the four required components, i.e., ClientDriver, JobTracker, FileServer, and Worker can be built. Please do *not* use other names of these four java files, as they will be referred to in the scripts we run to evaluate your code. You can also add more files and modify the content of Makefile if necessary. However, you should *not* modify any script files in the script folder.

The input argument of the four components is given below:
JobTracker <zookeeper serer>:<zookeeper port>
FileServer <zookeeper serer>:<zookeeper port> <dictionary file>
Worker <zookeeper serer>:<zookeeper port>
ClientDriver: <zookeeper server>:<zookeeper port> job/status <password hash>

As you can see above, the ClientDriver have two different functions: submitting a new job or checking its status. In other words, there are no persistent ClientDriver in the system. The ClientDriver should be able to submit a new job (and then quit, leaving the job running), and check its status at time later.

Once your implementation is finished, you can use the scripts we provide in the “script” folder to launch each component and evaluate your code. We will use these scripts to mark your

submissions as well. We show how to use these scripts below. Notice that the output of your code should meet the requirements of some scripts, otherwise your code may not be corrected evaluated. The input argument of JobTracker, FileServer, Worker, and ClientDriver can also be inferred from the these scripts.

1. Job Tracker start script:
   start_jobtracker.sh <zookeeper host> <zookeeper port>
2. Worker start script:
   start_worker.sh <zookeeper host> <zookeeper port>
3. File Server start script:
   start_fileserver.sh <zookeeper host> <zookeeper port>
4. Client submit job script
   submit_job.sh <zookeeper host> <zookeeper port> <password hash>
   This allows a user to submit a job request from the ClientDriver. It prints to the console if the job has been submitted successfully
5. Client check job status script
   check_job_status.sh <zookeeper host> <zookeeper port> <password hash>
   This allows a user to check the job status with the ClientDriver. It prints the status. There are three possible status types in this assignment:
   1. In Progress
   2. Password found: <password>
   3. Failed: <reason>
      reason can be:
      a. Password not found
      b. Failed to complete job (this is for cases when the password searching job can not finish due to reasons such as all workers failure)
      c. Job not found

Apparently, the first scripts will run the JobTracker, Worker, and FileServer, and the last two both run the ClientDriver to submit a new job and check its status, respectively.

To run these scripts, you may need to set them to be executable (chmod +x *.sh) after downloading and decompressing them from the portal.

A ZooKeeper service is expected to be running before running these scripts, and then you can evaluate your code manually by using the five scripts given above. To mark your submission, we will use a diffenrt script *exerciser.py* to run various tests. To give you a sense of this process, currently we provide this script with some simple tests which you can use to evaluate your code before the submission. Notice that the *exerciser* also require a running ZooKeeper service.

To run the exerciser script, you need to invoke the script from the same folder as the scripts mentioned above. This script assumes the ZooKeeper is running at port 8267. To the run the

script. you can do: ‘python exerciser.py local’. To run components on some other server, you can specify the hostname of IP address of the server in the command line argument, such as: ‘python exerciser.py ug49’ (components and tests will run on ug49).

The tests that the *exerciser* script will run include:
test 0: check if the required executables exist before running the test
test 1: check to see if all the components can be properly started and all the processes are running
test 2: submit a job and wait for the result to come back and check the client formats
test 3: inject one failure randomly in one of the JobTracker, FileServer or Workers, perform a validity check as test 2
test 4: inject several failure randomly in the JobTracker, FileServer or Workers, perform a validity check as test 2

The *exerciser* script you get only contains the first three tests.

P.S. Currently, the exerciser script can not work with the path with space. Please avoid putting the files in some folder with spaces in its (absolute) path.

FAQ:

1. What should I do if I can not decompress the tarball file of the lab4 scripts downloaded from the portal?
   It’s a weird problem that may happen if you use Chrome to download the tarball (especially on Linux). You may try to use some other browsers to download it.