ECE419 – Lab 4 Design Document

Shane Hu 999013632
Andrew Louis 998101977

1. Introduction

The proposed design for the dictionary attack system consists of 5 major components: ZooKeeper service, ClientDriver, JobTracker service, FileServer service, and a Worker pool. The ZooKeeper service acts as a lookup service and coordinates the communication between the other components of the design, handling leader election, group membership, and failure detection. The ClientDriver interfaces with the user and submits new job requests and job status requests to the JobTracker service. The JobTracker service manages the jobs submitted by the ClientDriver and divides each job into smaller tasks for Workers to complete. The job divisions correspond to a partition of the dictionary. The Workers then complete the tasks (they request the dictionary partition from the FileServer corresponding to a particular task), and then update the job's data field in Zookeeper if a match is found.

Our system successfully passes all tests in the **exerciser.py** script as well.

2. Task Management

When a JobTracker receives a job request from the ClientDriver, the job is divided up into 10 tasks. To communicate the tasks to the Workers, the JobTracker creates a new znode in Zookeeper under the "/jobs" znode with the job hash as the znode name. The string "~:10" is also added as the data for the znode as it is created. In this znode, 10 subnodes are also created.

Workers use ZooKeeper to listen for changes in the "/jobs" znode, when a new node is created, Workers randomly retrieves one of the subnodes under the new hash znode. The subnodes are numbered from 1-10, corresponding to the dictionary partition id. The Worker randomly picks and chooses a particular task indicated by the partition id and hash.

Once complete it updates the data string under the hash znode by decrementing the number after the colon (indicating task complete) and if a match is found, the "~" is replaced by the matched password.

When the JobTracker receives a job status request, it fetches the data string contained in the job hash znode. If the part of the string before the colon

contains a password, the JobTracker returns the password. Otherwise it checks the number after the colon. If the number is greater than 0, it returns job in progress, otherwise it returns password not found.

3. Components

3.1. ZooKeeper

The first level routes are: */jobs, /jobtracker, and /workers*, and */fileserver.*

The ClientDriver receives jobs from user-input, and adds them as children nodes of the jobs znode, such as: */jobs/<MD5 Hash>*.

A job's data field will be initialized with the string: "~:<*Number of Job Partitions*>". This data field will maintain the state of a job. If all subtasks are cleared by the Worker Pool without finding a match, the ending state will be "~:0", and similary if a match is found the job will be updated as: "<*password*>:<*number of unprocessed tasks*>". A job is initialized with children znodes that represent assignable tasks; the children are named from 1 to the number of job partitions, which is 10 by default.

The JobTracker and FileServer znodes maintain members maintain the listening addresses and membership of the JobTracker and FileServer services.

The workers znode maintains the hostnames of the members of the worker pool as its children. They are sequential-ephemeral znodes.

3.2. Client Driver

The ClientDriver is a simple client which takes user command line inputs to communicate the appropriate request to the JobTracker through Java sockets. The ClientDriver uses ZooKeeper to check if a JobTracker service is currently running, and if so, its address. It the JobTracker does not exist, the ClientDriver will watch the znode associated with the JobTracker service and wait until the service is created. The ClientDriver also watches the JobTracker znodes for failures. In case of a failures, the ClientDriver halts its progress and waits until another JobTracker to take over, before re-sending its requests.

3.3. JobTracker

The JobTracker service is started with an ephemeral znode in ZooKeeper. Backup services watch this znode for failure and will take over in the case of a failure. When a backup JobTracker service takes over from another failed instance, it

checks to see if there are any incomplete progress (such as incomplete task division) and completes them before accepting requests from ClientDrivers

3.4. FileServer

The FileServer reads the entire dictionary into memory, and holds it in N partitions, where N is the number of job partitions.

It consists of a server socket loop that spawns threads to handle connections as they are received.

The Worker sends the FileServer its dictionary division number required for it to complete its job, and the FileServer sends the corresponding dictionary partition, and the thread is terminated.

The FileServer's fault-tolerance mechanism is similar to that of the JobTracker service.

3.4. Worker Pool

A worker continuously polls ZooKeeper for the remaining jobs. It randomly selects a job that is in progress, and then randomly selects one of the job's sub-tasks. This is done to reduce the possibility of Workers processing the same task, and in addition, to ensure that the loss of a Worker does not stall jobs, thereby making our system more tolerant to loss.

After polling (and processing if there are any available jobs), the Worker sleeps for a small amount of time - again, to reduce the chance of overlapping tasks being processed.

On completion, the worker deletes the corresponding znode, and decrements the data field in the job znode.