School of Computer Science, McGill University

COMP-512 Distributed Systems, Fall 2020

Project 3: ZooKeeper & Distributed Computing

Due date: November 22 @ 6PM, Demo dates November 23/24/25

In this project, you will develop a simple distributed computing platform following a master-worker(s) architecture that uses ZooKeeper for coordination. The high-level objective of the platform is that a client can submit a "task/job" for computation to the platform, which will execute the task/job using one of its many workers and provide the client with the result. You are provided with a template code with some of the functionalities already implemented. For the purpose of this project, we will not consider crashes and failovers of the member processes and assume that all processes gracefully terminate.

Please ensure that you have watched the ZooKeeper lecture, gone over the tutorial, familiarized yourself with the ZookeeperSetup-Distributed documentation and set up your ZooKeeper ensemble, and done some sanity checks on it before starting the work on the project deliverable. Some useful references are given at the end of this document. If you have any additional questions, please use the mycourses discussion forum for project 3.

Onetime setup of ZK nodes

Using the ZooKeeper client CLI (`zkCli.sh`), create a permanent znode `/distXX` where `XX` is your group number (e.g. group 4 would create `/dist04`). Also create permanent znodes `/distXX/tasks` and `/distXX/workers`. You may need to create additional permanent znodes as needed based on your final design.

Setting up the Template

The `zk.tar` file provided with this description already has a substantial amount of code for your project. You can untar this into some directory in one of the SOCS servers.

```
zk
|-- clnt
|   |-- compileclnt.sh
|   |-- DistClient.java
|    -- runclnt.sh
|-- dist
|   |-- compilesrvr.sh
```

```
|   |-- DistProcess.java
|    -- runsrvr.sh
 -- task
    |-- compiletask.sh
    |-- DistTask.java
     -- MCPi.java
```

- Edit the `*.java` and `*.sh` files to replace `XX` with your group number where present. Also remember to replace the ZK server names and client ports with what you had used for your ZK ensemble setup.
- Compile the `task`, `clnt`, `dist` directories in that order. Remember to set your `ZOOBINDIR` environment variable and execute the env script before compiling (see the setup documentation for this).

The Current Application

- The `task` directory contains the definitions of "Jobs" classes that the client programs can construct an object and (serialize) submit to the distributed platform through the ZooKeeper. For the purpose of this project, we have a Monte Carlo based computation of the value of `pi` defined in `MCPi.java`.
- The `clnt` directory contains the client java program (`DistClient.java`) that creates the "job" object and submits it to the platform, awaits the result and retrieves it. It does this by creating a sequential znode `/distXX/tasks/task-` that contains the "job" object (serialized) as the data and then waiting for a `result` znode to be created under it.
- You can invoke the client in the following way (make sure that the `ZOOBINDIR` is set and the env script is executed.)

  ```
  $ ./runclnt.sh 500000
  ```

  Where `500000` is the number of samples (points) that the Monte Carlo simulation should use (the larger the number, the more computation it needs to perform, but the more accurate the computed value of `pi` would be).
- Ideally, you should not have to edit any of the java files in `task` or `clnt` directories, other than for any additional debugging that you think you may need.
- The `dist` directory contains `DistProcess.java` that defines the program code that all the members in the distributed platform have. This is where most of your work would be.
- A member process can be started as follows.

  ```
  $ ./runsrvr.sh
  ```

The general idea is that you start a process from a different machine.(And therefore, will have multiple members in the distributed platform).

- In the current implementation, if you start a `DistProcess` it tries to become the master by creating an ephemeral znode `/distXX/master`. A second process will detect that there is already a master, but does nothing else as of now.
- The master process listens for any child znodes under `/distXX/tasks`. It picks up these znodes, unserializes the data to build the "job" object and performs the "computation". It then serializes the job object and writes it (creates) back to `result` znode that is the child of the task znode. The master is also currently doing this inside its callback function, which was we discussed in the class is not the best place as it blocks the ZK client library.

Question 1: Implementation (80 Points)

1. The first change is to make sure that any additional `DisProcess` that you start after you have a master will become a "worker". You will have to figure out an extension to the current znode hierarchy (the ZK model) used by the application to accomplish this.
2. The master should be enhanced to keep track of the workers in the platform (as new workers join).
3. The master, instead of doing the computations itself (current setup) will assign the jobs to an idle worker.
4. A worker will take care of executing only one job at a given point in time. After it is done with its computation, it will go back to being idle. You will have to figure out a way of keeping track of who is idle/busy using ZK so that master can manage the assignment of jobs to the workers effectively.
5. If there are no idle workers but there are more jobs, the master will have to wait till a worker becomes free to assign the next job. Ideally we would like to prioritize the job that was added first, but we do not have to be strict about this.
6. There are some `TODO` labels throughout `DistProcesss.java` which should help you navigate to some of the locations where you need to do the above code changes. However, keep in mind that depending on your design and approach, you might have add extra code/functions.

A few things to keep in mind.

- All communication between the member processes of the distributed computing platform, including the clients of the platform will be through the ZooKeeper ensemble. I.e., by creating/removing znodes, updating the data component of the znode, etc.
- Depending on your design, you may have to create some other additional znodes, etc., over and above the general outline given above to accomplish this project.

- Make sure to put plenty of messages (print statements) throughout the various important steps of the code so that we can observe what each process is doing just by observing their display messages. Some examples are already present in the code given to you.

Question 2: Report (20 Points)

Write a short (2 pages **maximum**) report that mostly describes the ZooKeeper model that you designed (similar to the tree diagram in class). Explanation as to what each znode is for (including the type of the znodes), who does what, when, where. Also a short description of the contribution of each team member.

The report is due by **6 PM on November 22** to give the us enough time to read before the demos.

Demos

To grade your implementation we will use online demonstrations where you show your running system to us. It is recommended that your demonstration include a very short (less than 4 slides) presentation with highlights of your system and your implementation strategies.

The demonstrations of all groups will take place over three days (November 23/24/25). A sign-up sheet will be put in place so that you can reserve a time-slot. Show up early and have your system running, using a different set of machines for the ZK ensemble and for your client and DistProcess servers (You can run multiple clients from the same machine if you would like to). We would like to see at least 3 workers, 4 (or more) client jobs being submitted at the same time. Expect questions!

All code is due on MyCourses by the time of your demo.

Assumptions to Simplify the Project

- You do not have to worry about a worker having to switch to master when the master shutdown.
- It is assumed that the platform is shut down (i.e. kill all the processes) only when there are no pending client jobs.
- You can assume that a worker shuts down only when the entire platform is shutdown.

Some Useful Documentation/References

1. ZooKeeper TextBook (Oreilly) Chapters 2, 3. If you glance over some of the sections, they discuss the nuances of master-worker(s) implementations (We are not building such a complicated setup though).
2. ZooKeeper Java API doc. `https://zookeeper.apache.org/doc/current/`

3. ZooKeeper main APIs (create, get, etc.,). https://zookeeper.apache.org/doc/current/apidocs/zookeeper-server/org/apache/zookeeper/ZooKeeper.html

4. Link to All ZK packages and classes https://zookeeper.apache.org/doc/r3.6.2/apidocs/zookeeper-server/index.html

5. AsyncCallback interfaces (to receive asynchronous callback) https://zookeeper.apache.org/doc/current/apidocs/zookeeper-server/org/apache/zookeeper/AsyncCallback.html

6. Watcher Related https://zookeeper.apache.org/doc/current/apidocs/zookeeper-server/org/apache/zookeeper/Watcher.html
https://zookeeper.apache.org/doc/current/apidocs/zookeeper-server/org/apache/zookeeper/WatchedEvent.html