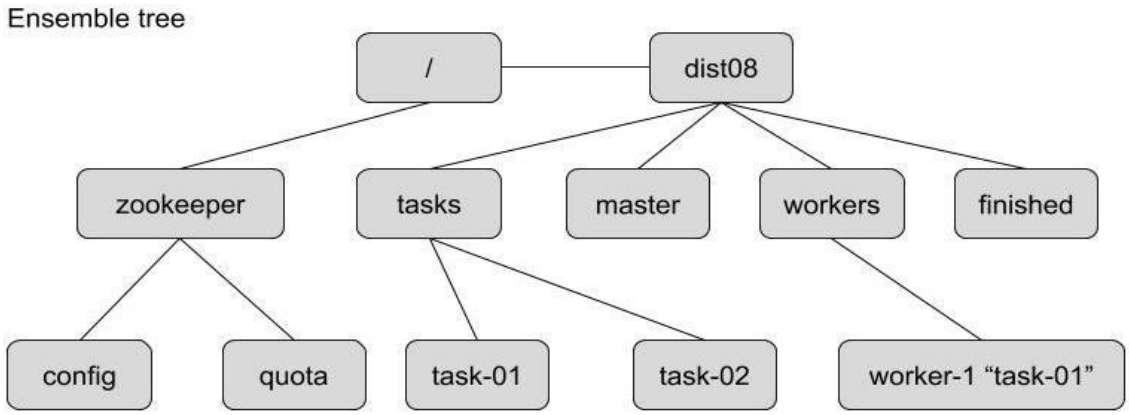
COMP-512 Distributed Systems, Fall 2020

Project 3: Zookeeper & Distributed Computing

Group8: Zhang Yong, Eric Shen, Winnie Tan

Design

In our design, aside from permanent znode */dist08/tasks* and */dist08/workers*, there is also another permanent znode */dist08/finished* used to denote which worker has finished a task, thus becoming idle again. All the communication between the zookeeper nodes is done by setting the data of a node, creating and deleting some flag znodes (children of */dist08/finished* in this implementation). A figure of the tree diagram of a snapshot of the program has been shown below (the names and the data of znodes are not exactly the same).



Functionalities

Worker

The worker znode in our design is a type of ephemeral znode, so it would be removed automatically if the process crashes. When a worker process gets started, a corresponding znode would also be created as well. At the very beginning, the data of the worker znode would be set to "idle" to denote its status and the name of it would be just a concatenation of "worker" and a random but unique identifier (e.g., */dist08/workers/worker-1*). Meanwhile, this worker process would install a watcher to its own worker znode. It will get notified when there is a *NodeDataChanged* event.

Whenever a task is available, the master process would try to set the data of an idle worker znode to be the task name. After that, when the worker process receives the event, it could just read the data and start to do the computation. Once it finished its job, it would create an ephemeral znode under */dist08/finished* (e.g., */dist08/finished/worker-1*) according to its worker identifier to let the master process know that it has been idle. After that, it would change its data back to "idle".

By taking advantage of changing the data of the worker znode and creating a new ephemeral znode, the sign of being busy and idle could be achieved elegantly.

Master

The master process would create an ephemeral znode *as /dist08/master* in the tree diagram. After that, it would keep track of the tasks, objects, and worker process by installing watches for both of them. Thus, whenever a new worker or task comes in, the master process would get notified and perform the corresponding actions. To speed up the performance, instead of querying the worker process availability when a new task arrives every time, the master has maintained a local hashmap to keep track of which worker is busy. This hashmap would be updated when the related worker znode changes. For instance, if there is a new task requested by the client, the master process would be notified and try to find an idle worker process and set the data of the worker

znode to the name of the specific task (e.g., */dist08/workers/worker-1* with data as "task-1").

The master process also installs a watch for the children of /*dist08/finished* znode. By doing so, it could know exactly when a worker process has done its job. Once a worker is done and a znode is created under */dist08/finished* znode, the master process would update its worker hashmap again and delete the newly created znode under */dist08/finished* (e.g., */dist08/finished/worker-1*) to denote that the master process has known worker-1 is idle again. However, there is a situation where there are no idle workers left and a new task arrives. To handle this, a new thread is created to try to wait for any idle worker process. Thus, the zookeeper library would not be blocked but is always ready to accept the new requests.

Also, before assigning an idle worker to a task, the master would only take the ones that are brand-new into consideration. For instance, if a task has been assigned to a worker but the worker is still working on the task, this task is guaranteed that it would not receive another worker to do the computation. This is achieved by maintaining a list of tasks assigned with workers (both finished and assigned but not finished yet). If a task is in this list, then it would not receive any more workers.

Others

In addition to the above znodes, */dist08/finished*, a permanent znode, acts as a holder to denote which worker has become available again. The reason why it is permanent is that just like workers and tasks, it could be regarded as a container for different children. Details for its use has been discussed in the two sections above.

For permanent *tasks* znode, it holds children which are the real tasks that the workers would perform when they are idle. Each time a client submits a task, a particular task would be created under it. After a worker finished the computation, it would create a persistent *result* znode under this specific task znode with the response stored in the data part. Both of them would be deleted when a task is finished.

To make the whole system handle more requests within a given time, asynchronized callback is adopted instead of synchronized callback as much as possible. This is because some of the zookeeper operations might take a relatively long time and the zookeeper client library is single threaded, a synchronized function might block the whole callback. Thus, by avoiding the synchronized API, the subsequent call-backs could be responded to more quickly.

Example

To better illustrate the control flow of the whole program, a tiny example is given below:

Assume there are a master and a worker *(/dist08/workers/worker-000001*) already active in the system. A client just gets started and is ready to submit a task. Since the master process has already installed a watch on the children of the */dist08/tasks*, the newly created */dist08/tasks/task-00000001* would trigger the process function in the master process. Then, the master would check whether or not this task has been completed by a worker. After finding out it is a totally new task, it would loop over the workers' hashmap to try to find an idle one. Since there is only one worker, the master would set the data of */dist08/workers/worker-000001* to be "task-00000001". At this time, the worker process would get notified since the data of its znode has been changed. Thus, it would start a new thread, grab the data value, "task-00000001", in this case, and try to get the *DistTask* object by querying *getData* on znode */dist08/tasks/task-00000001*. Then, the worker process would activate the computation step and do the job (e.g., create the result znode under the original task znode). Once it finishes, just before ending the callback, it would create a znode */dist08/finished/worker-000001* to denote it has finished and change its status back to "idle". This operation would trigger the process function on the master process again. Then, the master process would get the children of */dist08/finished*, delete */dist08/finished/worker-000001*, and update the status of *worker-000001* in its own worker hashmap.

Finally, the client would get notified as there is a result znode under the task znode. Thus, it completes the whole task handling process.

Contribution

Yong Zhang and Eric Shen are responsible for implementing the whole zookeeper system.
Winnie Tan is responsible for proposing a general design, editing the report, and testing.