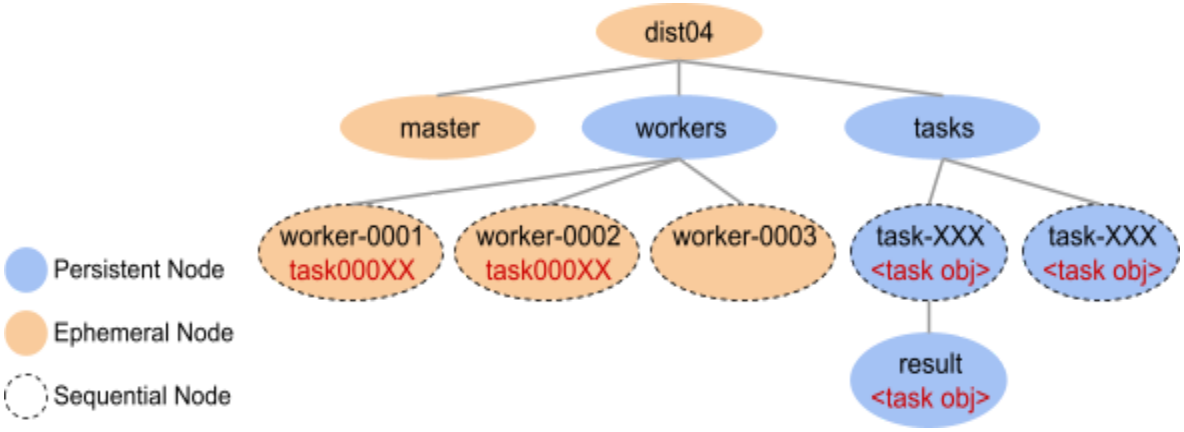
School of Computer Science, McGill University

COMP-512 Distributed Systems

Project 3: ZooKeeper & Distributed Computing

Group04: Elvin Zhang - Jack Hu - Mian Hamza

ZooKeeper Model



When we start a master, we create an ephemeral node "*/dist04/master*". When we start a worker, we create a sequential ephemeral node for that worker as a child of "*/dist04/workers*". Clients create sequential persistent task nodes as children of "*/dist04/tasks*", with the task objects as data. When a master assigns a task to a worker, it writes the task name to the worker's data field. After a worker completes a task, it creates a result node as a child as the task node with the result as data. And the corresponding client detects the result node, prints the result and deletes the task node.

Master

Initialization

The master Znode is a distributed process that interacts with Zookeeper using Java Zookeeper APIs. When we start up a server, and there's no master node under "*/dist04*", the server should function as a master and create itself as an ephemeral node "*/dist04/master*". This is because the master keeps track of the incoming tasks submitted by any Zookeeper client processes and afterwards when the master process shuts down it will just remove itself from the tree.

In this architecture, when a master process first starts up it's also in charge of performing bootstrapping for the Zookeeper programming model. In our design, the distributed architecture is set up such that there exists a top level node, "*/dist04*", then followed by it's children namely "*/dist04/tasks*", "*/dist04/workers*". Thus, during bootstrapping, the master would also create these (permenent) nodes if it doesn't already exist within. Lastly it would create itself within this tree structure as the "*dist04/master*" znode.

Keeping Track of Tasks and Results

The master installs a watcher and a call-back for changes to the children of "*/dist04/tasks*" znode in order to keep track of the incoming client tasks. In master, we also maintain an *OrderedSet pendingTasks* storing tasks that need to be assigned, and a *HashMap assignedTasks* recording which task has been assigned to which worker. When the master detects a change to the children of "*/dist04/tasks*", it adds new tasks' names (that are not contained in both *pendingTasks* and *assignedTasks*) into *pendingTasks*, and removes tasks from *assignedTasks* whose nodes are deleted(namely they are already finished). And then if there are idle workers, assign pending tasks. The assignment process will be discussed later.

The master also sets a watcher for changes to the children of each task znode "*/dist04/tasks/task-XXXXX*". When the result node under the znode is added by a worker, the master knows the task is finished. Thus, the master can change the worker' state from busy to idle, and assign another pending task.

Keeping Track of Workers

The master process also subscribes to the changes to the children of "/*dist04/workers*" as well in order to keep track of the number of available workers in the system (as workers join and leave the system). It has its own respective watcher and callback as well. The master also maintains a *HashSet* for *idleWorkers* and a *HashSet* for *busyWorkers*. When a new worker node is detected, the master adds it into *idleWorkers*. When the master assigns a task to a worker, it removes the worker from *idleWorkers*, and adds it into *busyWorkers*. Besides, when the master detects a new result node under a task node, it removes the worker from *busyWorkers*, and adds it into *idleWorkers*.

Task Assignment

Three conditions can trigger the task assignment process: 1) changes of workers are detected, which means there might be new workers joining that can be assigned tasks; 2) changes of tasks are detected, which means there might be new tasks created by clients that need to be assigned; 3) a new result node is detected, which means a task has been finished by a worker, and the worker becomes available for another pending task.

In the task assignment process, we iterate tasks in the *OrderedSet pendingTasks*. (It is defined as *OrderedSet* in order to assign tasks in the order that they are submitted by clients.) When there's still an idle worker, we assign the task to the worker by writing the task name "*task-XXXXX*" to the worker's data field. Otherwise, stop the iteration.

Worker

Initialization

The worker process is a distributed process that interacts with Zookeeper using the Java APIs as well. When a server process starts up, if there is not a master node, the server should function as a worker and then it creates an sequential-ephemeral znode under "*/dist04/workers*". The Zookeeper programming model would add a unique ID to each worker name at the end in order to keep all workers unique and easily differentiable in the system.

Task Processing

The workers in the system would wait (stay idle) until their data fields contain a task assigned to them by the master. The master would keep track of which workers are available/idle by setting a data watcher to its own znode. If a change to its data is detected, then it knows that a task is assigned. Then the worker starts a new thread and executes that task in that thread in order to not block the main event thread. When the computation is finished it would create a persistent "*/result*" znode under the node of the specific task that was assigned to it. This would inform the master as well as the client which submits this task that the task has been finished.

Client

The client Zookeeper process is in charge of serializing their tasks and submitting them to the tasks znode. It would create a persistent sequential znode under tasks. This way, multiple connected clients can create unique tasks for the system to intercept. Each respective client would also install a watcher and call-back on their own specific child "/*task-XXXXX*" znode in order to retrieve the result of the computation that was submitted for processing. When a worker has created the "*/result*" znode under a specific task, it shows the result and deletes the node "*/tasks/task-XXXXX*". Lastly, when a client disconnects, it also cleans up all of it's task znodes.

Contribution

Elvin Zhang: Designed the Zookeeper model and the architecture. Implemented the functionalities of master and workers.

Jack Hu: Designed and created a version of the Zookeeper distributed programming model with multiple workers and clients in the system. Created bash scripts for easy compilation and setup of the Zookeeper environment. Tested tasks assigned to multiple workers, various kinds of watcher and call-back functionality that subscribes to changes under the /tasks & and worker/ znode. Created bootstrapping and cleanup functionalities for the Zookeeper distributed system. You may checkout my branch here at: https://github.com/jhu960213/COMP_512_Projects/tree/dev_jack

Mian Hamza: Implemented a version of the Zookeeper functionality and architecture. Debugged the working Zookeeper model afterwards, including adding new functionality/modifying the architecture to circumvent errors.