Dig Data Management Assignment 4

信科四 郑元嘉 1800920541

Implement Mutex based on Zookeeper and Python

In this task, we're about to implement a Mutex (Mutual Exclusive) based on Zookeeper APIs on Python.
Here shows the steps.

1. Installation

Install Zookeeper by following the tutorial provided by TA.

2. Start the zookeeper

```
# /usr/local/zookeeper/zookeeper-3.4.14
$ ./bin/zkServer.sh [start-foreground/start]
```

3. Python Zookeeper API library - `kazoo`

We use library `kazoo` on Python to manipulate zookeeper.
Some essentail segments would be expanded on below. (Complete source code is append at *5. Complete Source Code*)

- Connect to zookeeper service and end it in the end.

  ```
  zk = KazooClient(hosts='127.0.0.1:2181')
  zk.start()
  zk.stop()
  ```

  Note that the port of zookeeper service is defined in `(/usr/local/zookeeper/zookeeper-3.4.14/)conf/zoo.cfg` file with a default of 2181.
- Ensure the node path before creating a node

  ```
  def ensure(self):
  ```

```
2    self.zk.ensure_path(self.parent_path)
```

It's easy to ignore this step and we'll find we're not allowed to create a node with its parent path not existing.

- Create a sequence node to indicate a key

```
1  new_node_path = self.zk.create(os.path.join(self.parent_path, self.node_pat
2  name = new_node_path.split('/')[-1]
3  self.name = name
```

- Ask of the minimal sequence number

```
1  nodes = sorted(self.zk.get_children(self.parent_path))
2  min_node = nodes[0]
3  if min_node != name:
4      print('task {} is awaiting...'.format(name))
5      self.t_0 = time.time()
6      min_node_path = os.path.join(self.parent_path, min_node)
7      self.zk.get(min_node_path, watch=self.task)
8  else:
9      self.task()
10 while not self.done:
11     pass
```

If it's the one with minimal sequence number, then start its task right away. If not, set a watch to the precedent node, then set a watch to the precedent node and the task will be kicked off when the watch sends the event.
Here's a trick, because the watch doesn't hold the program, we must apply while loop (or any similiar other) to wait for that the watch triggers the task and task is well done.

- Delete the node to delete the mutext

```
1  self.zk.delete(new_node_path)
```

Then, the metux of the task is totally gone through.

4. Experiments and Results

To simulate the multiple programs share the metux to run their own tasks, we run two processes at a time in the experiment. Plus we use `time.sleep()` and `random.random()` to mock the running time of tasks.
Here are the results:

Program 1

```
File/hadoopHw/hw4  master ✗
▶ python3 mutex.py
task task-0000000000 starts
task task-0000000000 ends
task task-0000000000 consumes 3.33 s
task task-0000000002 is awaiting...
task task-0000000002 starts
task task-0000000002 ends
task task-0000000002 consumes 0.13 s
task task-0000000004 is awaiting...
task task-0000000004 starts
task task-0000000004 ends
task task-0000000004 consumes 3.55 s
task task-0000000006 is awaiting...
task task-0000000006 starts
task task-0000000006 ends
task task-0000000006 consumes 4.67 s
task task-0000000008 is awaiting...
task task-0000000008 starts
task task-0000000008 ends
task task-0000000008 consumes 2.32 s
```

Program 2

```
File/hadoopHw/hw4  master ✗
▶ python3 mutex.py
task task-0000000001 is awaiting...
task task-0000000001 starts
task task-0000000001 ends
task task-0000000001 consumes 2.09 s
task task-0000000003 is awaiting...
task task-0000000003 starts
task task-0000000003 ends
task task-0000000003 consumes 2.89 s
task task-0000000005 is awaiting...
task task-0000000005 starts
task task-0000000005 ends
task task-0000000005 consumes 2.65 s
task task-0000000007 is awaiting...
task task-0000000007 starts
task task-0000000007 ends
task task-0000000007 consumes 4.84 s
task task-0000000009 is awaiting...
task task-0000000009 starts
task task-0000000009 ends
task task-0000000009 consumes 4.48 s
```

As seen, the two programs actually await when the shared mutex is occupied.

5. Complete Source Code

```python
from kazoo.client import KazooClient
import time
import random
import os



PARENT_PATH = '/mutex'
NODE_PATH = 'task-'



class Task:
    def __init__(self, zk, parent_path, node_path):
        self.zk = zk
        self.parent_path = parent_path
        self.node_path = node_path
        self.name = None
        self.done = False


    def ensure(self):
        self.zk.ensure_path(self.parent_path)


    def task(self, event=None):
        print('task {} starts'.format(self.name))
        t_s = time.time()
        time.sleep(random.random() * 5)
        print('task {} ends'.format(self.name))
        t_e = time.time()

        self.done = True
        print('task {} consumes {:.2f} s'.format(self.name, t_e - t_s))


    def run(self):
        self.ensure()
        new_node_path = self.zk.create(os.path.join(self.parent_path, self.node
        name = new_node_path.split('/')[-1]
        self.name = name
        nodes = sorted(self.zk.get_children(self.parent_path))
        min_node = nodes[0]
        if min_node != name:
            print('task {} is awaiting...'.format(name))
            self.t_0 = time.time()
```

```
47              precedent_node_path = os.path.join(self.parent_path, nodes[nodes.in
48              self.zk.get(precedent_node_path, watch=self.task)
49          else:
50              self.task()
51          while not self.done:
52              pass
53
54          self.zk.delete(new_node_path)
55
56
57
58  if __name__ == '__main__':
59      zk = KazooClient(hosts='127.0.0.1:2181')
60      zk.start()
61
62      for i in range(5):
63          task = Task(zk, PARENT_PATH, NODE_PATH)
64          task.run()
65
66      zk.stop()
```