Distributed Systems and Cloud Computing (CLOUDS) 2014

Laboratory on Zookeeper
May 22, 2014

Prof. Marko Vukolić

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. http://creativecommons.org/licenses/by-sa/3.0/
**General Information**

Work in groups of two. Refer to the Zookeeper lecture slides during the Lab.

Your hand-in should contain:
- **your student (login) id** (or that of your pair) you used in the exercises and your names.
- answers to questions in Exercises 1-4, and
- .java files and logs for Exercises 5 and 6.

Deadline for the hand-in (exclusively by email to vukolic AT eurecom.fr) is **May 29, 2014**.

**0 Setup (Please read carefully)**

0.1. The Lab material is available on [http://www.eurecom.fr/~vukolic/clouds/](http://www.eurecom.fr/~vukolic/clouds/) and contains:
- DSCC2014 – Zookeeper Lab.pdf (this file)
- zookeeper-3.4.6.tar.gz (Zookeeper server and client v3.4.6)
- zkLab.tar.gz (Eclipse project/source files for the Lab)

0.2. Download all three files into a new directory nested in your /homes. Unpack the zookeeper-3.4.6.tar.gz archive to a directory of your choice (installation directory)

0.3. Zookeeper client is started (relatively from the installation directory) as follows

`> bin/zkCli.sh –server motisma:2181`

after few moments you should see the following prompt

```
[zk: motisma:2181(CONNECTED) 0]
```

**Your Zookeeper client is now up and running and ready for use.**

0.4. Enter:

```
[zk: motisma:2181(CONNECTED) 0] help
```

for an overview of Zookeper command line interface (CLI) syntax. Most of the commands should be familiar from the Zookeeper lecture.

0.5. **Create your first znode with path /<your login id>.** Unless specified differently, use always this path as the parent node for exercises. Example:

```
[zk: motisma:2181(CONNECTED) 1] create /vukolic this_is_znode_data
```
PART A: Getting to know Zookeeper

1 Ephemeral vs. Regular (a.k.a. Persistent) znodes

1.1. (Init) Create the root znode for this exercise /<your login id>/ex1

Example:

```
[zk: motisma:2181(CONNECTED) 2] create /vukolic/ex1 ex1
```

Two znodes that we created so far are regular (also known as persistent) zNodes

1.2. (Create Persistent) Create another persistent znode (e.g., /<your login id>/ex1/persistent)

1.3. (Create Ephemeral) Now create one ephemeral znode (e.g., /<your login id>/ex1/ephemeral) This can be done by inserting the –e option between create and the path (as in create –e <path> <data>). If you forget <data> you will crash the CLI client (bug).

1.4. (List/getChildren) List the children of /<your login id>/ex1 using the ls command in the CLI (ls in CLI is getChildren in the API).

1.5. (Test Ephemeral) Now quit the client by typing

```
[zk: motisma:2181(CONNECTED) 6] quit
```

Then restart the client and repeat 1.4.

Explain your observations in 1.4 and 1.5 and explain the differences.

1.6. (Ephemeral children) Repeat step 1.3 and then try to create a child of this node (e.g., /<your login id>/ex1/ephemeral/child). What happens?
2 Sequential flag

2.1. **(Init)** Create the root znode for this exercise /<your login id>/ex2

2.2. **(Sequential basics)** Create a few znodes with SEQUENTIAL flag (simply called sequential znodes) as children of /<your login id>/ex2. Try option –s in create, e.g.,

```
[zk: motisma:2181(CONNECTED) 3] create -s /vukolic/ex2/child someData
```

Answer the following questions:

2.2.1. How is the name of a sequential znode constructed? How long is the suffix?
2.2.2. Is it possible to create persistent sequential znodes?
2.2.3. Is it possible to create ephemeral sequential znodes?
2.2.4. Create now few non-sequential znodes and then delete few znodes. Create again few sequential znodes. Explain how suffix numbers are generated.
2.2.5. Assume that after executing 2.1 you created 100 sequential children of /<your login id>/ex2, followed by 5 non sequential children of /<your login id>/ex2, followed by deletion of 10 most recent sequential child znodes. If you’d create another sequential node, what would be its suffix?

2.3. **(Scope of sequence numbers)** Are sequence numbers of sequential children of a znode (e.g., /<your login id>/ex2) and sequence numbers of sequential children of one of its children (e.g., /<your login id>/ex2/abc) related?

2.4. **(Sequence numbers across multiple clients)** Are sequence numbers unique to a given client? To answer this question, you may want to start two (or more) clients in separate windows (or on separate machines), both creating sequential znodes as children of /<your login id>/ex2

**Tip.** If you’d like to start over, use the `rmr` CLI command to delete a znode including its children. Example:

```
[zk: motisma:2181(CONNECTED) 28] rmr /vukolic/ex2
```
3 Watches

All of the read operations in ZooKeeper (which operations are these?) have the option of setting a watch as a side effect. The goal of this exercise is to understand the behavior of watches in Zookeeper and to give you ideas how watches can be used in coordinating distributed systems.

3.1. **(Init)** Create the root znode for this exercise with path:

```
<rootex3> = /<your login id>/ex3
```

Start the second client in parallel. In one client you will set the watches while the other makes actions that (might) trigger watches.

3.2. **(First watch)** In Client 1, do a get on `<rootex3>` and set a watch. For example

```
[zk: motisma:2181(CONNECTED) 0] get /vukolic/ex3 true
```

Then with Client 2 modify `<rootex3>`.

```
[zk: motisma:2181(CONNECTED) 35] set /vukolic/ex3 newData
```

What happens in Client 1?

3.3. **(Durability of watches)** Building upon 3.2., modify again `<rootex3>` in Client 2.

What happens in Client 1? How many times can a watch be triggered in Zookeeper?

3.4. **(More watches)** `getChildren` command can also set a watch. Do this in Client 1 for `<rootex3>`. Remember in CLI `getChildren` is called `ls`.

Now, just like in 3.3, modify `<rootex3>` in Client 2. Does this trigger the watch at Client 1? Try to explain why.

3.5. **(Child watches)** In Client 2 create a child node of our `<rootex3>`. What happens in Client 1? Create another child of our root in Client 2. What happens now in Client 1?

3.6. **(Watch types summary)** Summarizing 3.3-3.6., explain the difference between child watches and data watchers. Which command sets which watch?

3.7. **(Child watches and nesting)** In this exercise we test nested child watchers. In Client 1, create another child of our `<rootex3>`. Set child watches on both newly created child and `<rootex3>`. 
Now, in Client 2, create a grandchild (a child of a child) of <rootex3>. How many watches are triggered in Client 1?

Create yet another child of <rootex3> in Client 2. Does this trigger the watch in Client 1? Do grandchildren trigger child watches in Zookeeper?

3.8. **(Watches and znode deletion)** Create a node /<your login id>/ex3/deletetest (called testnode)

Make sure you have three clients running in parallel.

- In Client 1, set a child watch on testnode.
- In Client 2, set a data watch on testnode.
- Finally, in Client 3, delete the testnode.

Which watches are triggered by a delete?

3.9. **More on watches:**

- Watches are maintained locally at the ZooKeeper server to which the client is connected.

- A client will never see a change for which it has set a watch until it first sees the watch event.

- The order of watch events from ZooKeeper corresponds to the order of the updates as seen by the ZooKeeper service.

- Due to asynchrony and network delays, different clients may see watches triggered at different times. **The time when a watch triggers cannot be used as synchronization method itself.**
PART B: Programming Zookeeper

Eclipse project setup

i) Open Eclipse. Go to Java perspective.

ii) Import the Zookeeper lab project into Workspace as follows
   a. File > Import
   b. General > Existing Projects into Workspace. Click Next.
   c. Select archive file > Browse
   d. Find zkLab.tar.gz at the place you downloaded it. Click Open.
   e. Click Finish

iii) You should see the ZookeeperLab Java project with the following src files
      
      package fr.eurecom.rs.clouds.ZookeeperLab
      Consensus.java
      ConsensusTest.java
      SyncPrimitive.java
      ZKUtils.java
      
      package fr.eurecom.rs.clouds.ZookeeperLab.watches
      Configuration.java
      ConfigurationTest.java

iv) Make sure your console is open: Window > Show View > Console
4. Consensus

4.1. **(Understand the Consensus implementation)**

The Java project you just set up contains an implementation of the Consensus synchronization primitive. The implementation follows the pseudocode given in slide 47 in the Zookeeper lecture notes.

**Take time** to look at the code and understand the implementation. Compare it to the pseudocode of slide 47. At this moment, you need to understand only the following 4 classes from the `fr.eurecom.rs.clouds.ZookeeperLab` package:

a. Consensus.java  
b. ConsensusTest.java  
c. SyncPrimitive.java  
d. ZKUtils.java

Refer to internal javadoc and code comments for brief descriptions.

4.2. **(Run consensus)**

a. Open ConsensusTest.java and put your consensus proposal value in the `myProp` variable.  
b. Run ConsensusTest.java. See if your proposal ``wins`` consensus.  
c. Answer the following questions:  
   4.2.1. What is the purpose of the 2\textsuperscript{nd} parameter in the Consensus() constructor?  
   4.2.2. How is the proposal zNode constructed?  
   4.2.3. Would changing the 3\textsuperscript{rd} parameter in the Consensus() constructor impact the decision of consensus?  
   4.2.4. Suggest a **single line code modification (not addition) in ConsensusTest.java** that would ensure you (with very high probability) to ``win'' consensus.  
   4.2.5. Implement and run the change you suggested in 4.2.4

4.3. **(Understanding Consensus assumptions)**

4.3.1. Suggest what could a malicious client do to violate consensus properties?  
4.3.2. Which property of consensus (refer to slide 47) could be violated with the attack you suggested in 4.3.1?  

**Note:** Observe that the exercise title mentions “understanding” not “attacking”. Do not implement the attack.
5. **Group Membership**

Now it is your turn to program Zookeeper.

The goal is to implement a Group Membership service for which the pseudocode is given in slide 49 of the Zookeeper lecture notes.

The goal is that, much like in the Consensus example, you come up with GroupMemb.java and GroupMembTest.java

**Tip:** Feel free to, if you want, make a separate copy of Consensus.java and ConsensusTest.java, rename them and edit to obtain GroupMemb.java and GroupMembTest.java. Feel free to reuse SyncPrimitive.java and ZKUtils.java.

Your implementation should adhere to several **requirements**:

i) Use “/group” as the root path for synchronization (just like in slide 49).

ii) Your hostname (`ZKUtils.getHostName()`) and your login id (`System.getProperty("user.name")`) should be nicely concatenated (e.g., using a dash “-“) and serve as a name (again, referring to name from slide 49)

iii) Test client (GroupMembTest.java) should register a client to the group membership service and periodically log membership to a file (note here that you **do not** have to set watches to achieve this, although you might). Test client should run the Group Membership service for at least 15 mins without disconnecting from the server to fill the log.

5.1. **Hand-in for this exercise** should contain

- GroupMemb.java (implementation of the group membership service)
- GroupMembTest.java (test client of the group membership service)
- A log described in requirement iii) above (only in case you finish the experiment during the lab hours).

5.2. **(Bonus, optional)**

Try to implement the membership service without access to unique names (i.e., without access to hostname and your student login ID).

**Tip:** Zookeeper Javadoc shows up in Eclipse, but you may also prefer an HTML version on

http://zookeeper.apache.org/doc/r3.4.6/api/overview-summary.html
6. Watches – Configuration service

In this exercise we program very simple watches.

The goal of the exercise is to modify the Configuration.java to obtain the configuration service as the one on slide 48 in the lecture notes.

To simplify the exercise, Configuration.java is already given as a stub. Just replace the TODOs with functional code, following the pseudocode of slide 48.

Requirements: We assume that clients know the znode configuration path (“/config”). However, it might be the case that this znode does not exist at the time configuration service is instantiated. The configuration service should just read from Zookeeper, writes are done externally (see also Test below).

Test: To test your implementation, once happy with Configuration.java, run ConfigurationTest.java, and then run a CLI client and try

```
[zk: motisma:2181(CONNECTED) 0] create /config brandNewConfig
```

or

```
[zk: motisma:2181(CONNECTED) 0] set /config evenNewerConfig
```

Tip: You do not have to modify ConfigurationTest.java. If you modify Configuration.java properly, the ConfigurationTest.java should be sufficient to perform the test without modifications.

Your hand-in for this exercise should contain:
- Modified Configuration.java
- Log of the configuration service obtained after running ConfigurationTest.java and CLI client as described above
7. **Zookeeper server to go (optional)**

In case you need to finish the exercise outside of lab hours with no access to *motisma*, you may want to deploy your own Zookeeper server(s).

Deploying a non-replicated (and hence for development purposes only) Zookeeper server is straightforward. Relatively from the Zookeeper installation directory:

a. rename `conf/zoo_sample.cfg` to `conf/zoo.cfg`
b. Run
   i. `bin/zkServer.sh start` (Linux), or
   ii. `bin/zkServer.cmd` (Windows)

that’s it. You should now have a server running on *localhost*, port 2181.

More information is available at