ID2203 –Distributed Systems, Advanced
Course Project – VT21 P3

TA – Max Meldrum <mmeldrum@kth.se>, Harald Ng <hng@kth.se>

1 Organisation

The course project consists of 4 parts. The first part is simply an introduction to Kompics and is optional if you have worked with Kompics before. The rest will be summarised in a final project report which is graded at the end of the course and forms the basis for the lab part of the course.
To motivate you to start working in time, there is an intermediate milestone after task 2.1. You will have to submit a preliminary report, describing what has been done so far and the plan for the rest of the project, in Canvas. Every student will be assigned another student to review their preliminary report and write a short review with feedback, which will also be submitted in Canvas, as well as discussed during the peer feedback session.
Hand in both the report (as .pdf) and the source code (as .zip or .tar.gz) of your project in Canvas. Do not package up the report with the code, but submit them as separate files!
Furthermore, it is strongly recommended to use GIT for managing your source code. You are free to use public Github or KTH's gits[1] repository. If you do use either, please provide a link to the repository in your report. (But still submit the code in Canvas anyway, to have a reference snapshot for grading.)

1.1 Dates

Preliminary Report 23 February in Canvas

Preliminary Report Peer Feedback Session 26 February Lab exercise

Final Report 12 March in Canvas

1.2 Goals

The goal of the project is to implement and test and simple partitioned, distributed in-memory key-value store with linearisable operation semantics. You have significant freedoms in your choice of implementation, but you will need to motivate all your decisions in the report. Some of the issues to consider are:

[1] https://gits-15.sys.kth.se/

- Networking Protocol
- Bootstrapping
- Group Membership
- Failure Detector
- Routing Protocol
- Replication Algorithm

You will also have to write verification scenarios for your implementation. Distributed algorithms are notoriously difficult to test and verify, so do not take this part of the tasks lightly.

1.3 Requirements

For this lab you will need the following:

- Java SDK, version 8 or newer. Oracle Java is recommended but OpenJDK should work as well.
- Maven or SBT
- An IDE like Netbeans, Eclipse, IntelliJ is recommended, but any simple text-editor would be enough.

1.4 Time

Be sure to plan enough time for the project. The project will require a significant amount of code, and testing distributed systems is notoriously difficult and time intensive. It is recommended that you start as early as possibly to get a feel for Kompics and how long it takes to implement something in it.

2 Tasks

2.0 Introduction to Kompics (0 Points)

Implement all the *PingPong* examples from the Kompics tutorial at:
`https://kompics.github.io/kompics-scala/tutorial` and/or complete *Programming Exercise 1* in Canvas.
This task is optional and does not give any points. If you have questions you can ask them during the exercise sessions.

2.1 Infrastructure (15 Points)

For this task you have to design the basic infrastructural layers for the key-value store. Your system should support a key-space of some kind (e.g., strings or integers), that is partitionable (e.g., via consistent hashing or static/dynamic range-assignment). It is sufficient for this variant of the project to support a single partition that is redundant,

such that all values are replicated with a specific replication degree δ. You are free to keep δ as a configuration value or hardcode it, as long as it fulfils the requirements of your chosen replication algorithm (task 2.2), so plan ahead.
For this task you need to be able to set up the system, assign nodes to form a replication group, lookup (preloaded) values[2] from an external system (client), and detect (but not necessarily handle) failures of nodes. Additionally, you should be able to broadcast information within the replication group.
On the verification side you have to write simulator scenarios that test at least two of the features mentioned above. You should also be able to run in a simple deployment (it's fine to run multiple JVMs on the same machine and consider them separate nodes). For the report describe and motivate all your decisions and tests.

Note Since not all of the subtasks of this section are of particular interest to this course, we are providing a template project you can use as a starting point for your code. You can find it at:

- https://gits-15.sys.kth.se/mmeldrum/id2203project21

You are not required to use all or even any of the code in there, it is merely provided as a convenience to avoid that people waste too much time on unrelated coding work.

2.2 KV-Store (15 Points)

After you have gotten the basic infrastructure working, you have to add a $PUT(key, value)$ operation, that updates (or adds) a value at a key, to the GET from the previous task. As mentioned in the goals, the operations should fulfil the linearisable property, so make sure choose you the right replication algorithm.
As before, be sure to write test scenarios for the simulator that check the correctness of your implementation. Especially be very careful to explain how you are verifying the linearisability of the store.

2.3 Compare-and-Swap (10 Bonus Points)

For bonus points, also implement a compare-and-swap (CAS[3]) operation that compares the current value at the key to a given reference value and only updates with the new value if the old value and the reference value are the same.
Try to reuse your test scenarios from task 2.2.

[2] This is effectively a $GET(key) : value$ operation.
[3] $CAS(key, referenceValue, newValue) \rightarrow oldValue$