ID2203 –Distributed Systems, Advanced
Course Project – VT16 P3

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 in Canvas. Every group will be assigned another group to review their preliminary report and write a short review with feedback, which will also be submitted in Canvas.

The project is to be done in pairs, with clearly divided responsibilities. It is important to point out the contributions of each member in the final report.

Hand in both the report and the source code of your project in Canvas.
Also it is strongly recommended to git for managing your source code. You are free to use public Github or KTH's gits repository. If you do use either, please provide a link to the repository in your report.

1.1 Dates

Tutorial Session January 31st

Preliminary Report February 19th in Canvas

Preliminary Report Peer Feedback Session February 22nd during the Exercise

Code Working March 2nd for the Project session

Final Report March 12th 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:

- 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.

You are free to write your project in either *Java* or *Scala* Kompics.

Note: A good labour distribution for the tasks might be to have one group member work on the algorithm implementations and the other one on the test scenarios after having agreed on the general architecture and interfaces.

1.3 Requirements

For this lab you will need the following:

- Java SDK, version 7 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:
http://kompics.sics.se and/or complete *Programming Exercise 1* in Canvas.
This task is optional and does not give any points. However, if you haven't worked with Kompics before you should most definitely do it. If you plan on doing the project in Java prioritise the PingPong, if you plan on doing it in Scala, rather to the Programming Exercise first.
It is recommended (but not required), that you continue to do the Programming Exercises in Canvas as the course progresses, as they contain helpful information that you can use for your own solutions.
If you have questions you can ask them during the tutorial exercise session.

2.1 Infrastructure (10 Points)

For this task you have to design the basic infrastructural layers for the key-value store. Your system should support a partitioned key-space of some kind (e.g. hash-partitioned strings or range-partitioned integers). The partitions need to be distributed over the available nodes such that all value 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 partitions and replication groups, lookup (preloaded) values[1] from an external system (client), and detect (but not necessarily handle) failures of nodes. Additionally, you should be able to broadcast information within replication groups and possibly across.
On the verification side you have to write simulator scenarios that test all 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/lkroll/id2203project17. 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 (20 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

[1] This is effectively a $GET(key) : value$ operation.

sure choose you the right replication algorithm.
For more points, also implement a compare-and-swap ($CAS(key, referenceValue, newValue) : success?$) 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.
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 Reconfiguration (10 Points + 10 Bonus Points)

At this point your store is fairly static and can't really deal with node failures apart from complaining about them. For this task you should implement reconfiguration support for your replication groups and your routing protocol. You should be able to deal with both nodes leaving the system (treat a voluntary leave the same way as a failure, for simplicity) and new nodes joining the system. There are many ways to interpret the semantics of this, including making some of the partitions unavailable while they are under-replicated. Any approach you take is acceptable as long as you document it properly in the report. However, you have to make sure that reconfiguration does not violate linearisability for *correct* nodes.
To get full points for this task you'll have to write test scenarios that reconfigure the system and verify that for all correct nodes the operations are still linearisable.

Note This task is fairly open ended and can get quite difficult, depending on the choices you have made before. For the 10 required points, you are expected to make a good effort in writing some code and demonstrate in the report that you have understood what is involved in a proper implementation of this. For actually getting it to work correctly, you are awarded 10 bonus points.