ID2203 –Distributed Systems, Advanced
Course Project – VT23 P3

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

1 Introduction

The course project aims to give hands-on experience in working with distributed systems. The project emphasizes applying what you learned from the lectures and verifying that your implementation is correct. You will submit a final report that summarizes and highlights your work including problem description, design, solution and testing, etc. 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 Structure and Dates

The projects will be conducted in **groups of 2-3 people**. Each group writes one report and presents their work orally together.

Final Report March 17 in Canvas

Oral Presentation March 20, 21: 9:00-13:00 (Preliminary, Doodle TBA)

1.2 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 Rust and how long it takes to implement something in it. You will also have to write verification scenarios for your implementation. As seen in section 3, the testing plays a significant role for grading so do not take this part of the tasks lightly.

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

2 Topics

We provide a range of suggested project topics involving Omni-Paxos, but you are also allowed to propose your own topic as long as it aligns with the course contents (see section 2.4). In any case, you must submit a project proposal in Canvas.

2.1 Powering Open Source Systems with Omni-Paxos

Omni-Paxos is a novel replicated state machine protocol with increased resilience and performance opportunities compared to the current state-of-the-art protocols such as Raft. This topic aims at replacing Raft with Omni-Paxos in one of the following open source systems:

- ToyDB
- TiKV

Each of the listed systems either has dependencies on Raft libraries or has its own Raft implementation. Pick one of the systems, remove the Raft dependency, and replace it with Omni-Paxos.

2.2 Benchmarking and Optimizing Omni-Paxos

In the lectures, we have mainly focused on the safety and liveness properties of Omni-Paxos. For practical usage, the system can be further enhanced. Pick one of the following features and implement it in `omnipaxos`.

2.2.1 Service layer for reconfiguration

We covered how Omni-Paxos can safely and efficiently perform a reconfiguration using the parallel log migration in the service layer. Your task is to implement the parallel log migration. A newly joining server must first receive the complete log before it can start its new omnipaxos instance.
Currently, the library supports stopping the current configuration by deciding a StopSign. Your task is to detect that a StopSign has been decided, migrate the log to new servers, and start the new OmniPaxos instance.
For full points, your solution should allow users to define an optional custom transmission scheme e.g. a newly joining server s_5 should only pull the log from servers s_1 and s_2.

2.2.2 Selective log synchronization

In Omni-Paxos, all followers send their log suffix to the leader with the <Promise> message in the prepare phase. The leader then adopts the most updated suffix among a majority of promises. Thus, the suffixes sent from the other, less updated, followers are redundant. Furthermore, the most updated suffix is only transferred from one follower to the leader.

To optimize the prepare phase and avoid redundant transfers, your task is to design and implement a selective log synchronization feature. This requires the leader to first get knowledge of which followers have the most updated log (or parts of it) and then request the different parts of the suffix from different followers in parallel.

Implement the described selective log synchronization in `omnipaxos_core`. In the written report, you must motivate why your design is correct.

2.2.3 Benchmarking: Omni-Paxos vs. Leaderless protocols

Omni-Paxos is an RSM based on Sequence Consensus which builds a log in strictly growing order. There is also a new family of *leaderless* protocols that are not leader-based and try to exploit operations that are commutative (non-conflicting) to obtain better performance.

In this project, you will investigate the pros and cons of Omni-Paxos in comparison to leaderless protocols. You will implement Omni-Paxos in the benchmarking framework fantoch and compare it to the leaderless protocols. In your report, discuss the following questions based on both qualitative and quantitative results:

- *How do the different protocols behave with low and high conflict rates?*
- *What are the benefits of considering commutative operations in RSMs (performance, practicality, fault-tolerance)? What are the restrictions?*
- *Would it be possible to extend Omni-Paxos to better handle commutative operations?*
- *How does the leaderless protocols handle partial connectivity?*

2.3 Building Application with Omni-Paxos

You can also choose to build your own application using omnipaxos. In these projects, you will have more freedom to pick which libraries and frameworks for the runtime, network, storage, etc.

2.3.1 Distributed Coordination Service

Replicated State Machines have received wide adoption in the form of distributed coordination services such as etcd[2] and ZooKeeper[3]. In this project, you will build such

[2]https://etcd.io/
[3]https://zookeeper.apache.org/

a service from scratch using omnipaxos. The solution should at least support basic operations such as reading and writing using a similar API to one of those systems (e.g., hierarchical key-value as in etcd or file system as in ZooKeeper).

2.3.2 Highly-Available Web Server

In this project, you will build a web server that is made fault-tolerant using omnipaxos. The web server should be accessed via REST or gRPC and store some data that is accessed in a consistent manner. Furthermore, let each node maintain some statistics on the number of requests they have handled, and make the node with the most requests take over leadership when it passes a certain threshold.

2.4 Propose Own Project

You are also free to propose your own project using your choice of language, framework or system. The project should be closely related to the course contents.
Note: If you propose your own project, make sure we have confirmed the proposal before you start working on it!

3 Grading

The maximum score for the project assignment is 40 points, with 20 points based on the implementation and 20 points based on the demo, testing, and quality of the written report. It is therefore equally important to focus on testing and writing a good report as coding!