Algoritmos e Sistemas Distribuídos- Project Phase 2

João Carlos Antunes Leitão

NOVA Laboratory for Computer Science and Informatics (NOVA LINCS)

and

Departamento de Informática

Faculdade de Ciências e Tecnologia

Universidade NOVA de Lisboa

Preliminary revised version

27^{th} November 2020

1 Overview

This document discusses the second phase of the ASD project for 2020/21. The project has two phases which are fully independent between them. The primary goal of the project is to build and compare two state machine replication [6] solutions (that will be applied to replicate a simple application whose state is essentially a hash map) that differ on the aggrement protocol being employed. One of them will use Paxos [3, 4, 5, 1] (but since multiple Paxos instances will be required, you will have to rely on some simple meta-protocol that wraps the different Paxos protocol) and Multi-Paxos. You are going to compare these two different design alternatives by conducting an experimental evaluation in a system with three replicas that are subjected to different operational load generated by different number of clients. Clients are going to be emulated by YCSB [2]. Your evaluation will observe the resulting throughput and latency as observed by clients. Notice that in the experiments we will aim at having read operations that have strong consistency semantincs, which means that they have to be executed in a slot of the state machine (and ordered using the aggreement protocol used in that solution, either Paxos or Multi-Paxos). While you are not required to evaluate the system under faults or when adding replicas to it, your implementations should be correct if any of these events happen.
In the following the target protocol architecture (and their interfaces) for solving this phase of the project are presented (Section 2). The programming environment for developing the project is briefly discussed next (Section 3). Finally, the document concludes by providing some information on the delivery rules for this phase of the project (Section 4).

2 Solution Architecture

Figure 1 illustrates the layering of protocols that you will have to have to for phase 1 of the project[1]. The figure also describes the interactions between the protocols, from which the **interfaces** of the protocols can be derived. These

[1]for evaluation purposes this layering is *mandatory*.

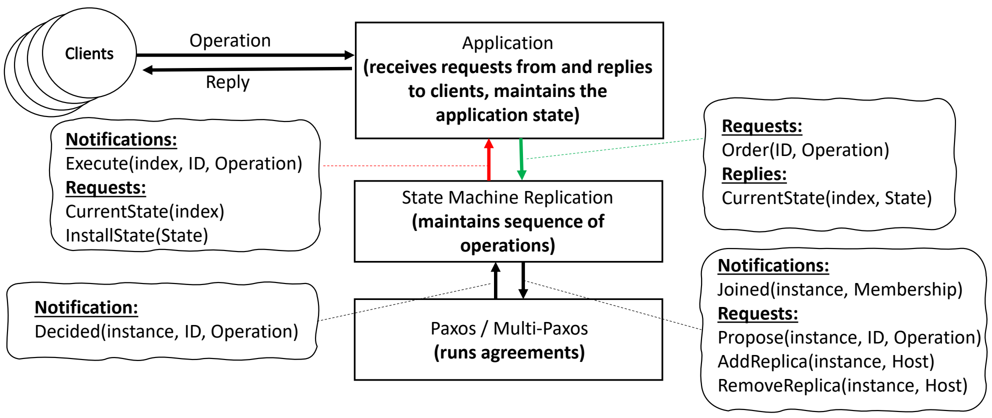
Clients
Operation
Reply
Application
(receives requests from and replies to clients, maintains the application state)
Notifications:
Execute(index, ID, Operation)
Requests:
CurrentState(index)
InstallState(State)
Requests:
Order(ID, Operation)
Replies:
CurrentState(index, State)
State Machine Replication
(maintains sequence of operations)
Notification:
Decided(instance, ID, Operation)
Paxos / Multi-Paxos
(runs agreements)
Notifications:
Joined(instance, Membership)
Requests:
Propose(instance, ID, Operation)
AddReplica(instance, Host)
RemoveReplica(instance, Host)

Figure 1: Architecture of a System Replica (and client interations).

interfaces must be respected by your implementations, and the java project that you will receive will already provide the Babel events that are used to materialize these interfaces.
Every protocol that you will implement should have an equivalent interface to communicate with the other local protocols (this is materialized by Requests and Indications). This is essential to ensure that you can swap one protocol implementation for another and that your process can still execute correctly. Notice that the messages exchanged by each protocol that you develop with the (equivalent) protocol on other processes is not fixed. You should model these messages following the specification of the protocols that you are implementing. Notice that if you careful in your implementation you can implement a single State Machine Replication protocol that is consistent with the use of either Paxos or Multi-Paxos.
Below the options for each of the relevant protocols is provided (with the references to papers when applicable).

2.1 *State Machine Replication*

This protocol is responsible for receiving the (client) operations from the application, and replicate them through the agreement protocol (i.e., Paxos or Multi-Paxos). It is the state machine that memorizes which operation has been decided for each position of the sequence of commands of the state machine and notifies the application to execute these operations (in the appropriate order).
The protocol is also responsible for managing the communication channel (i.e., open TCP connections), manage the membership of the system, which includes requesting to be added to the replica set when a new replica joins the system, and inform the agreement protocol of modifications in the membership. This implies thtat the state machine replication protocol is responsible also for detecting failures of replicas, attempting to reestablish connections with them a (configurable) number of times, and when suspecting that a replica having failed, issue an operation for the state machine to remove that replica.
When joining a replica set, it is necessary to copy the current state of the system when the new replica is added. This process, usually named *state transfer* is handled by the state machine replication protocol, that can request a copy of the current application state to the application, and upon receiving it, can send that state (and the information concerning the current position of the sequence of commands in which the replica was added) to the newly joining replica. To avoid all existing replicas to do this, one way you can address this is by having the replica that received the request of the new replica to be added to the system replying to the new replica with this information (notice that the State Machine Replication protocol needs to exchange messages through the network with other processes).
You can strive to use the same implementation of this protocol for the protocol stack with paxos and multi-paxos, although you might need to pass an aditional parameter to the state machine to inform of the protocol being used for agreement. This happens because when using Multi-Paxos, there is a notion of a leader (that the state machine needs to be notified about by the Multi-Paxos protocol), whereas when using Paxos this notion does not exists. In Multi-Paxos, since only the leader proposes commands to be decided, other replicas that receive requests from clients

must send these requests to the current leader, again at the state machine replication protocolo level. This implies that when the leader changes, requests received by clients that have not been ordered yet by the agreement protocol, must be resubmitted to the new leader.

2.2 Agreement Protocol

You will implement two different variants of aggrement protocol in this project: Paxos and Multi-Paxos. In your implementation avoid optimizations such as running multiple instances at the same time.
Paxos does not has a leader, and every process will propose commands to be decided across the different instances. You can decide what to do when a Paxos protocol receives a message for an instance that it has not yet started (i.e., where it has not received a proposal): you can reply following the protocol specification, or you can store the message locally and wait for the local state machine protocol to propose a value for that instance, and only after that process the message. Notice that you need to be aware of membership changes to be able to compute how many processes are a majority. The best suggestion for implementing Paxos, since you will need multiple instances, is to store the state of the Paxos algorithm within a Java class, and store multiple instances of this class in a map, where the key is the instance number. As such when you receive a message for a particular instance, you can fetch the current state of that instance in your map, and process the message by only modifying the state of Paxos for that instance. All Paxos messages have to be tagged with the instance for which they refer (instances can be a monotonic integer).
Multi-Paxos implementations should follow the specification presented in the Lectures. Notice however that in this case you will be required to answer to instances without having recieved a local initial proposal. This is because when using Multi-Paxos, only the leader proposes commands. In this protocol you also have to be careful to track the membership of the system, since you must know which replicas are part of the system in each instance to be able to compute the majority.
Notice that in both protocols, a process should not participate in instances that occur when the replica is not part of the system (i.e., before being added to the system or after being removed from the system).

2.3 Application

The application layer is going to be provided to you. It will have the code necessary to support all interactions to clients and it will use the prescribed interface to interact with the state machine replication layer.
The application can also expose its internal state (in a serialized form) and install a copy of state gathered from another replica. This is relevant to allow a new replica of the application to be added to the system while the system is operating.

2.4 Clients (YCSB)

To inject load in the system (i.e., execute operations) we are going to use the popular YCSB system. We will provide a driver that allows for YCSB to interact with the replicated application described above. Note that YCSB might send operations to any of the active replicas. Also YCSB executed multiple "client threads" where each one emulated an individual client. This is important because in the experiments we will want to study the differences between the latency (measured in mili-seconds) and throughput (measured in operations per second) as the total number of clients in the system increases. YCSB already computes and outputs both of these metrics for you.
You might need to run more than one instance of YCSB such that you can distribute the load of clients across multiple machines (a single physical machine has limits to the number of client threads it can execute concurrenlty without the clients becomming bottlenecked. The evaluation should saturate the servers and not the clients, as that would yield incorrect experimental results. If you rely on multple instances of YCSB to run your experiments, you will need to (manually) combine the results outputed by each instance. To that end you should compute the average of the latency observed on each instance, and you should compute the sum of the throughput reported by each YCSB instance.

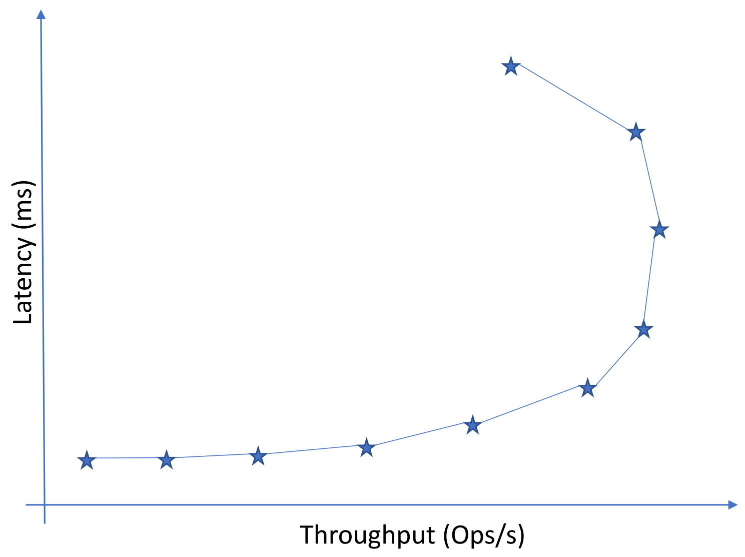

Figure 2: Fake throughput-latency plot represeting the performance of a single system.

2.5 Evaluation

You will conduct experiments to compare the latency and throughput of both solutions. To do that you will vary the total number of clients (threads across YCSB processes) issuing operations over the system. Since in our system read operations are being treated in the same way as write operations (i.e., they are ordered in the state machine) the workload can use any distribution of write and read operations. For simplicity and uniformity, lets use 50%W 50%R. When executing different number of clients interating with your replicated system (that will be composed of 3 replicas) YCSB will report both the latency of client operations and throughput (i.e., operations per second). The idea is that you plot the in a graph, where the x-axis reports the throughput of the system (if you use more than one YCSB instance, you should sum the throughput observed by each instance) and on the y-axis you report the latency (if you use more than one YCSB instance, you should present the average of the latency experienced by client threads in each YCSB instance). In your experiments be careful to avoid to saturate the clients, if you need to run more client threads than a single machine can handle you can run multiple YCSB instances across diferent machines simultaneously. In the plot you represent, for each system the data point for each experiment and you increase the number of clients and connect these sequentially with a line).
The plot described above, which is commonly refered as a throughput-latency plot, hides from the reader the number of client threads that are being reported. This information should be provided in the text on your report. Additionally, when looking at the plots, lines that are more to the right and bottom of the plot usually represent systems with better performance.
A mock-up of one such plot is presented in Figure 2 representing a single system (evidently the real plot should have numbers on it).

3 Programming Environment

The students will develop their project using the Java language (1.8 minimum). Development will be conducted using a framework developed in the context of the NOVA LINCS laboratory[2] written by Pedro Fouto, Pedro Ákos Cost, João

[2] `http://nova-lincs.di.fct.unl.pt`

Leitãoa, whose internal code-name is (still) **Babel**[3].
The framework resorts to the Netty framework[4] to support inter-process communication through sockets (although it was designed to hide this from the programmer). The framework will be discussed in the labs, and example protocols will be made available to students. Moreover the top layer that is responsible for injecting load in the system (i.e., propagate messages and receive them) is offered.
The javadoc of the framework can be (temporarily) found here: `https://asc.di.fct.unl.pt/~jleitao/babel/`.
The framework was specifically designed thinking about two complementary goals: i) quick design and implementation of efficient distributed protocols; and ii) teaching distributed algorithms in advanced courses. A significant effort was made to make it such that the code maps closely to protocols descriptions using (modern) pseudo-code. The goal is that you can focus on the key aspects of the protocols, their operation, and their correctness, and that you can easily implement and execute such protocols.
The course discussion board in Moodle can (and should be) used to ask for support in using or clarify any aspect on the operation of Babel.
While this is the second year that Babel is being used in this course, and the current version has also been used to develop several research prototypes, the framework itself is still considered a prototype, and naturally some bugs (or idiotic interfaces) can be found. We will address these as soon as possible. The development team is open to suggestions and comments. You can make such comments or improvement suggestions either through the Moodle discussion board or by e-mail to the course professor.

4 Operational Aspects and Delivery Rules

Group Formation

Groups should remain the same as in the first phase of the project.

The DI and NOVA LINCS research cluster

The technical specification of the cluster as well as the documentation on how to use it (including changing your group password and making reservations) is online at: `https://cluster.di.fct.unl.pt`. You should read the documentation carefully. The cluster is accessible from anywhere in the world through ssh. Be careful with password management. Never use a weak password to avoid attacks and intrusions on our infrastructure. Incorrect or irresponsible use of the department resources made available to students will be persecuted through disciplinary actions predicted in the School regulations.
All groups will have their quota in the cluster increased appropriately for the expected complexity of the project. Additional quota can be requested to the course professor.

Evaluation Criteria

The project delivery includes both the code and a written report that should have the format of a short paper. Considering that format, this document referes from this point onward as simply *paper*.
The paper should be at most 10 pages long, including all tables, figures, and references. It should be written with font size no bigger than 10pts and be in two column format. It is highly recommended (for your own sanity) that you use LaTeXto do this.
The paper must contain clear and readable pseudo-code for each of the implemented protocols, alongside a description of the intuition of these protocols. A correctness argument for protocol that was devised or adapted by students

[3]We are still not happy with the name – for different reasons – so we are still accepting suggestions, although we might submit a paper about this on early December.

[4]`https://netty.io`

will be positively considered in grading the project. The written report should also provide information about all experimental work conducted by the students to evaluate their solution in practice (i.e., description of experiments, setup, parameters) as well as the results and a discussion of those results.
The project will be evaluated by the correctness of the implemented solutions, its efficiency, and the quality of the implementations (in terms of code readability).
The quality and clearness of the report of the project will have an impact the final grade. Notice however that students with a poorly written report might (accidentally) penalized on the evaluation of the correctness the solutions employed. This phase of the project will be graded in a scale from 1 to 20.

Deadline

Delivery of phase 2 of the project is due on 13 December 2020 at 23:59:59.
The delivery shall be made by e-mail to the address jc.leitao@fct.unl.pt with a subject: "ASD Phase 2 Delivery - #student1 .. #studentn".
The e-mail should contain two attachments: i) a zip file with the project files, without libraries or build directories (include src, pom.xml, and any configuration file that you created) and ii) a pdf file with your written report. If you have an issue with sending the zip file you can do one of the following: a) put a password on the zip file that should be 'asd2020' with no quotation marks; or b) put the zip file in a google drive, and send the public access link.
Project deliveries that do not follow these guidelines precisely may not be evaluated (yielding an automatic grade of zero).

References

[1] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: An engineering perspective. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, page 398–407, New York, NY, USA, 2007. Association for Computing Machinery.

[2] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 143–154, New York, NY, USA, 2010. Association for Computing Machinery.

[3] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[4] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, December 2001.

[5] Robbert Van Renesse and Deniz Altinbuken. Paxos made moderately complex. *ACM Comput. Surv.*, 47(3), February 2015.

[6] Rubbert van Renesse. *State Machine Replication with Benign Failures*, page 83–102. Association for Computing Machinery, New York, NY, USA, 2019.