Algorithms and Distributed Systems — Project Phase 2

João Carlos Antunes Leitão, Alex Davidson

NOVA Laboratory for Computer Science and Informatics (NOVA LINCS)

and

Departamento de Informática

Faculdade de Ciências e Tecnologia

Universidade NOVA de Lisboa

Version 1.2

25th October 2024

1 Overview

This document discusses the second phase of the ASD project for 2024/2025. The project focuses on the study and implementation of a distributed system for managing the state of a key-value (KV) map. The KV map will be managed in two separate system models that you must implement:

- a State Machine Replication (SMR) algorithm [4], followed by a Multi-Paxos [3] Agreement protocol;
- an ABD Quorum [1] protocol.

In the case of SMR + Multi-Paxos, it is the SMR algorithm that manages the overall state and membership of the system. The SMR algorithm will use an underlying agreement protocol to decide consensus on the ordering of operations, via Multi-Paxos. In the case of ABD, there is a single protocol that manages the membership, state, and ordering of operations that are executed, via the application of the ABD agreement. In both cases, the agreement and membership functionality must use some sort of coherent broadcast mechanism for sending messages to other nodes in the system.

Overall goals. Overall, the goals/steps of the second phase of the project are defined as follows:

- Implement a SMR algorithm, that is agnostic to the choice of agreement protocol.
- Implement an agreement protocol based on Multi-Paxos.
- Implement an agreement protocol based on ABD Quorum.
- Conduct experimental analysis using 3 nodes, and using YCSB-based clients [2] to compare the system using both agreement protocols, comparing throughput and latency, as observed by clients.

Implementation and evaluation. 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 semantics, which means that they have to be executed in a slot of the state machine that is used (and ordered using the explicit agreement mechanism). 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.

Written report. Your implemented solution must be accompanied a report, using the latex template provided, that details the following information.

- The design and implementation of your system
- A performance analysis of your system, comparing your system using both agreement protocols.

The report must have a maximum of 8 pages, including figures and tables, but excluding bibliography.

Layout of rest of document. In the following, the target protocol architecture (and its interfaces) for solving this phase of the project will be presented (Section 2). Afterwards, the programming environment for developing the project is briefly discussed (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 2 of the project[1]. The figure also describes the interactions between the protocols, from which the **interfaces** of the protocols can be derived. These 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 instantiate these interfaces.

Every protocol that you will implement should have an equivalent interface to communicate with the other local protocols (this is instantiated by Requests and Indications). This should allow you to swap one agreement protocol implementation for another while your process still executes 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 protocol that you are implementing.

2.1 Key-Value map

The KV map is implemented in the code for the main application. The interface is standard, and you don't need to worry about including functionality for handling the standard operations (reads/writes), since these are already implemented at the application level.

2.2 State Machine Replication + Multi-Paxos Agreement

State Machine Replication protocol The State Machine Replication (SMR) protocol is responsible for receiving the (client) operations from the application, and replicate them through the agreement protocol. It

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

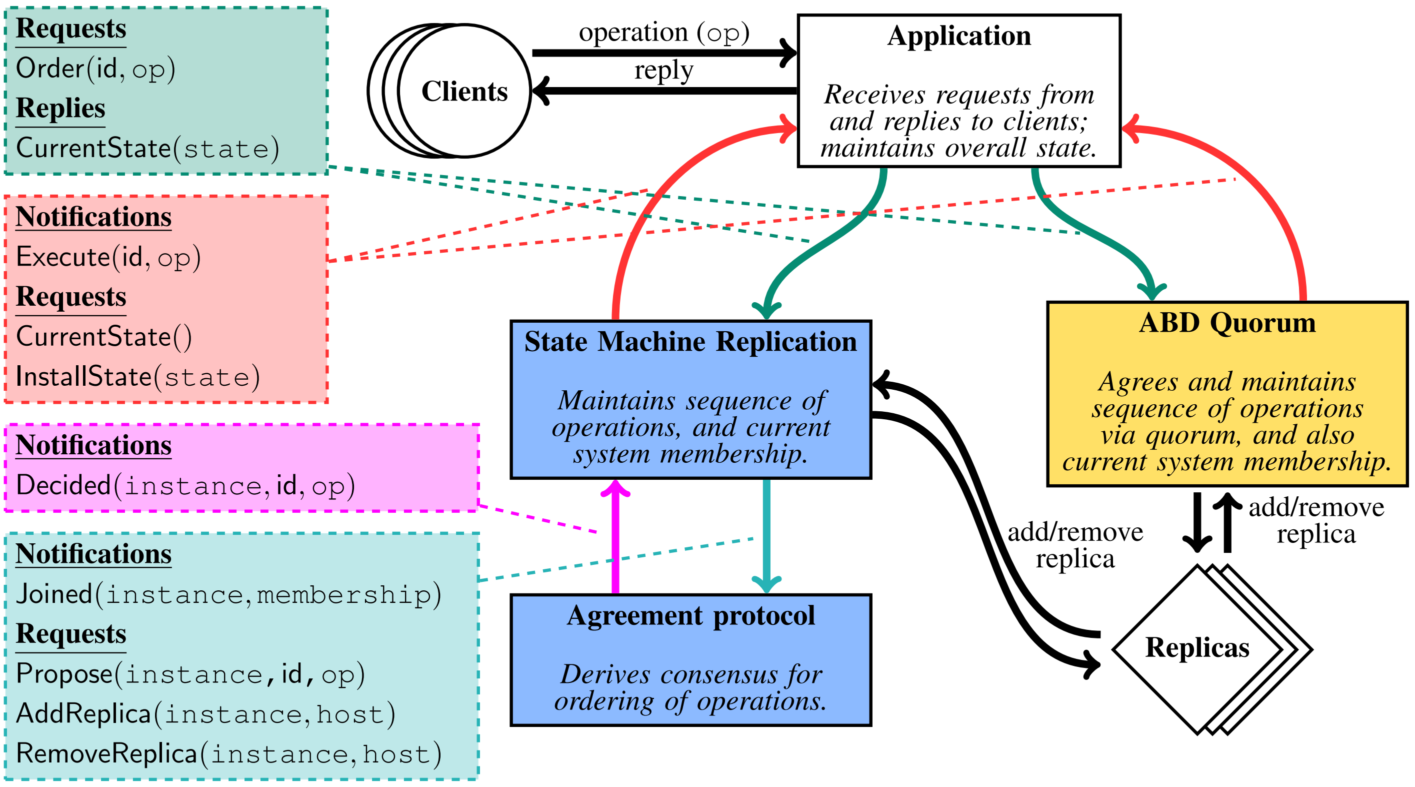
Requests
Order(id, op)
Replies
CurrentState(state)
Notifications
Execute(id, op)
Requests
CurrentState()
InstallState(state)
Notifications
Decided(instance, id, op)
Notifications
Joined(instance, membership)
Requests
Propose(instance, id, op)
AddReplica(instance, host)
RemoveReplica(instance, host)
Clients
operation (op)
reply
Application
Receives requests from and replies to clients; maintains overall state.
State Machine Replication
Maintains sequence of operations, and current system membership.
ABD Quorum
Agrees and maintains sequence of operations via quorum, and also current system membership.
Agreement protocol
Derives consensus for ordering of operations.
add/remove replica
add/remove replica
Replicas

Figure 1: Architecture of replicas (and client interactions) for both Multi-Paxos and ABD designs.

is the state machine that keeps track of 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), managing the membership of the system, which includes requesting to be added to the replica set when a new replica joins the system, and informing the agreement protocol of modifications in the membership. This implies that the state machine replication protocol is responsible also for detecting failures of replicas, attempting to re-establish connections with them a (configurable) number of times, and — when suspecting that a replica having failed — issuing an operation for the state machine to remove that replica.

State transfer. 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).

Independence from agreement protocol. Your implementation of this protocol should be, as much as possible, independent of the underlying agreement protocol. However, it is expected that you might need to pass an additional 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 ABD this notion does not exist. 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 protocol 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. When using ABD, whenever a new instance of the protocol is initialized (i.e., for each of the positions of the sequence of operations managed by the state machine) it might be required that the state machine protocol provides to the protocol the membership of the system.

Multi-Paxos agreement protocol. The Multi-Paxos agreement protocol will decide on the ordering of operations executed in the system. In your implementation avoid optimizations such as running multiple instances at the same time (as this requires a significant engineering effort to implement correctly).

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 received a local initial proposal. This is because when using Multi-Paxos, only the leader proposes commands. Therefore, 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 protocol 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. 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.

Finally, it should be noted that 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 ABD Quorum Alternative

You will implement an alternative to the SMR + Multi-Paxos setting, by using ABD Quorum to decide the ordering of all operations in the system. In this case, ABD handles both membership requests from other replicas, and standard operations to be inserted in the KV map. As such, ABD will replace the functionality of both the SMR and agreement components in the other system design.

For the ABD Quorum implementation, you can use any broadcast protocol (e.g. based on flood, or something more complex). The more resilient your broadcast protocol to potential failure cases, the better. Again, you will have choices that you can make regarding the way that messages are queued, and these are similar to those that are present in the Multi-Paxos implementation that you will write. Keeping track of membership for ensuring valid quorums will also be important.

The notion of instances may still be pertinent here, since you may have multiple invocations of the ABD protocol running at once.

2.4 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.

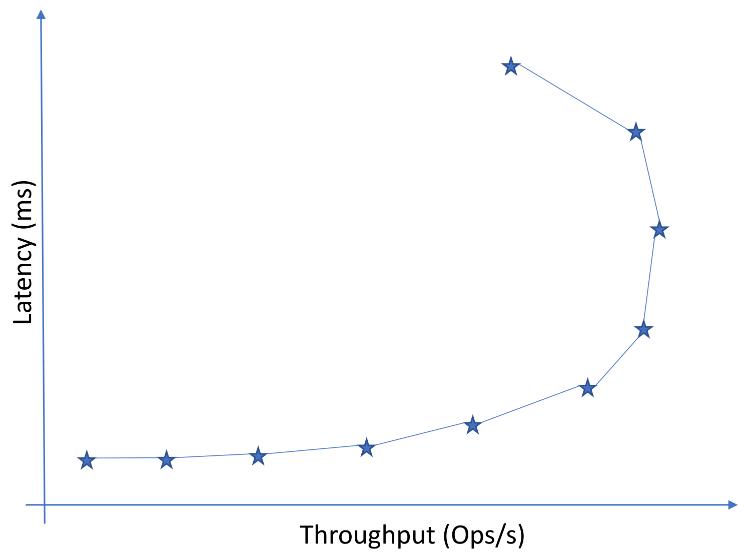

Figure 2: Fake throughput-latency plot representing the performance of a single system.

Handling the KV map in ABD. In Multi-Paxos, the consensus protocol is used to determine the order of operations that are pushed to the map. In ABD, things are slightly more nuanced. In particular, recall that the way we defined standard ABD was with respect to a single key-value pair. Note that we effectively need to define an ordering of operations for each individual key-value pair (think *serialisability*).

2.5 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 milli-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 concurrently without the clients becoming bottlenecked. The evaluation should saturate the servers and not the clients, as that would yield incorrect experimental results. If you rely on multiple instances of YCSB to run your experiments, you will need to (manually) combine the results outputted 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.

2.6 Performance Evaluation

You will conduct experiments to examine the latency and throughput of the solutions that you have implemented. To do that you will vary the total number of clients (threads across YCSB processes) issuing operations over the system. Since in your 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, you can use 50%W 50%R. When executing different number of clients interacting 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, and on the y-axis you report the latency (e.g. see Figure 2 for a mock example, where the real plot should have numbers on it). In your experiments, be careful to avoid saturating the clients, if you need to run more client threads than a single machine can handle you can run multiple YCSB instances across different machines simultaneously. If you do this, remember that the throughput from YCSB should be summed across all instances that are running, and the latency should be averaged.

The plot described above, which is commonly referred as a throughput-latency plot, hides from the reader the number of client threads that are being reported. This information **must** be provided in the text on your report. It is worth noting that, when looking at the plots, lines that are more to the right and bottom of the plot usually represent systems with better performance.

3 Programming Environment

The students will develop their project using the Java language (version 11 minimum). Development will be conducted using the **Babel** framework developed in the context of the NOVA LINCS laboratory[2] written by Pedro Fouto, Pedro Ákos Costa, João Leitão. The javadoc of the framework can be found here: `https://asc.di.fct.unl.pt/~jleitao/babel/`.

The framework resorts to the Netty framework[3] 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., propagating messages and receive them) is offered.

The course has a Discord channel (advertised previously over email) that can (and should be) used to ask for support in using or clarify any aspect on the operation of Babel.

4 Operational Aspects and Delivery Rules

Group Formation

Groups should remain the same as in the first phase of the project.[4]

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

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

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

[4] We note that it is permitted for existing groups to be broken, but existing groups cannot be combined nor augmented with new members.

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, as described 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 from 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. The paper should be at most 8 pages long, including all tables, figures, and references. It should be written with font size no bigger than 11pt 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 pseudocode 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 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 solution, its efficiency, and the quality of the implementations (in terms of code readability). The quality and clearness of the report will have an impact the final grade. Notice however that students with a poorly written report might be (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 29 November 2024 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:

1. a zip file with the project files, without libraries or build directories (include src, pom.xml, and any configuration file that you created);
2. 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 ’asd2425’ 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] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *J. ACM*, 42(1):124–142, January 1995.

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

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