# dida-tkvs

Design and Implementation of Distributed Applications 2024-2025
IST (MEIC-A / MEIC-T / METI)
Project

Version 1.1

## 1 Introduction

The goal of the dida-tkvs project is to implement a simple transactional key-value store that is replicated using the Paxos consensus algorithm. The application is composed of different components, namely:

- Multiple *servers*, that keep the (replicated) state of the key-value store.
- Multiple *transactional clients*, that execute update transactions in a loop.
- A single *console*, that can send configuration commands to the servers.

The project should be developed in different phases, described later in this document.

## 2 Operation of the **dida-tkvs**

The dida-tkvs stores key-value pairs. The key-value store associated to each value a *timestamp*, assigned when the value is written. To simplify the project, client can only execute a simple form of update transaction, with a fixed format. More precisely, ever transaction reads exactly two keys and writes in a single key.

To execute a transaction, the client operates as follows. It selects a key and sends a *read request* to the server(s), obtaining the most recent key value and timestamp. It then selects another key and repeats the process. After performing both reads, the client selects a key to update and a value to write on that key. Finally, the client sends a *commit request*, that includes the keys involved in the transaction, the timestamp of the values that have been read, and the value to be written. The servers coordinate to assign a unique write timestamp, higher than any previous timestamp, to the update. The transaction commits if, at the logical time defined by the write timestamp, the keys that have been read still have the version observed by the client; in that case, the update is applied. If one of the keys that have been read by the client has been updated, between the read operation and commit time (this can be verified, by checking if the current timestamp no longer matches the read timestamp), the transaction aborts and no value is written.

Clients send requests to all servers and return as soon as they receive a reply from one of the servers.

## 3 Servers, Replicas and Configurations

There are $n$ servers. Each server is identified by a number in the range $0, \ldots, (n-1)$. As a simplification, we assume that the number of servers is fixed to $n = 5$. To further simplify the project, it is assumed that the IP address and port of the server 0 is well known by all clients and servers (it is passed as a parameter when a process starts) and that the other servers run on consecutive ports in the same machine (i.e., if server 0 runs on port $y$, server 1 runs on port $y + 1$).

Also as a simplification, we will execute all processes (servers and clients) on the same machine. Naturally, a real system would run nodes in different machines.

We also assume that all servers are running when clients start. All servers always act as Paxos “learners”, i.e., they receive requests from clients, observe the order assigned to requests, execute the requests, and may reply to clients. However ,at a given point in time, only a subset of the servers may act as “proposers” and “acceptors”. This set of nodes is defined as a *configuration.* To simplify the project, we assume that there are only 3 possible configurations:

- *Configuration 0:* composed of servers $\{0, 1, 2\}$.
- *Configuration 1:* composed of servers $\{1, 2, 3\}$.
- *Configuration 2:* composed of servers $\{2, 3, 4\}$.

As an example, in *Configuration 0* nodes $\{0, 1, 2\}$ can be proposers, nodes $\{0, 1, 2\}$ are acceptors, and nodes $\{0, 1, 2, 3, 4\}$ are learners.

The system starts in *Configuration 0* and can dynamically change to the next configuration (up to configuration *Configuration 2*).

We assume that key 0 of the key-value store is reserved to keep the current configuration. *Transactional clients* should not be allowed to write in this key.

## 4 Programming Language and Background

The project should be implemented in Java and GRPC. IST students should have the necessary background to execute the project, as it builds on the material covered by the

Distributed Systems course at the BSc level. Students that are not familiar with Java and GRPC will need to learn the required background using self study and by doing the exercises prepared for the Distributed Systems course (that are available online).

## 5 Project Steps

It is recommended to develop the project in a stepwise manner. The following steps should be followed:

- *Step 0: Client-server implementation with uncoordinated servers.* The set of servers is fixed. In this step, clients send requests to servers that execute them as soon as they receive the request without any coordination. Naturally, the state of the servers may diverge if concurrent requests from different servers are received in different orders. A simple version of this code has been already implemented and provided to students.

- *Step 1: Fixed sequencer.* The set of servers is fixed. Servers receive the requests but do not execute them immediately. One of the replicas (the leader replica) assigns a sequence number to each request. All replicas execute request in the order specified by the leader. Note that this is similar to fast Paxos when there is a single leader.

- *Step 2: Paxos.* The set of servers is fixed. Servers receive the requests but do not execute them immediately. Servers run Paxos to order requests. Servers only start an instance of Paxos after the previous instance has terminated.

- *Step 3: Reconfigurable Paxos.* Same as above but with support for reconfiguration. It should be possible to replace one replica by another replica. Servers only start an instance of Paxos after the previous instance has terminated.

- *Step 4: Multi-Paxos.* Same as above but servers can initiate an instance of Paxos before the previous instance has terminated.

## 6 Console

The *console* is used to send commands to servers to control their behaviour. The console can issue the following commands:

- *setleader:* that permits to simulate an imperfect leader oracle; it can instruct multiple servers to act as leaders.

- *reconfigure:* that instructs server nodes to move to the next configuration. A reconfiguration request is treated as a transaction that updates key 0. The console advances configuration in the correct order (i.e., the system mas pass trough *Configuration 1* before going to *Configuration 2*.
- *setdebug (mode):* that activates pre-configured behaviours on a given server.

The following debug modes should be implemented. Students are free to implement other debug modes to test their algorithms.

- debugmode1 (crash): the server crashes (i.e., exits).
- debugmode2 (freeze): the server "freezes", i.e., blocks all requests from clients (except from the console) until it is "un-freezes".
- debugmode3 (un-freeze): the server "un-freezes", i.e., blocks all requests from clients (except from the console) until it is "un-freeze".
- debugmode4 (slow-mode-on): the server applies a random delay before it processes any request (except request from the console).
- debugmode5 (slow-mode-off): the server stops applying a random delay to requests.

## 7 Final Report

Students should prepare a final report describing the developed solution (max. 6 pages). In this report, students should follow the typical approach of a technical paper, first describing very briefly the problem they are going to solve, the proposed implementation solutions, and the relative advantages of each solution. Please avoid including in the report any information already mentioned in this project description. The report should include an explanation of the algorithms used and justifications for the design decisions. The final reports should be written using LaTeX. A template of the paper format will be provided to the students.

## 8 Checkpoint and Final Submission

The grading process includes an intermediate optional checkpoint and a mandatory final submission. For the checkpoint the students may submit a preliminary implementation of the project; if they do so, the checkpoint grade may improve their final grade. The goal of the checkpoint is to control the evolution of the implementation effort.

The final submission should include the source code (in electronic format) and the associated report (max. 6 pages).

## 9 Project Goals

The students can set their own goals for the project. Depending on the goals achieved, there is a ceiling to the top grade they can receive, as follows:

- *Vanilla:* Execute Step 1 by the checkpoint and Step 2 by the final delivery date. Max grade: 17/20.
- *Premium:* Execute Step 1 and Step 2 by the checkpoint and Step 3 by the final delivery date. Max grade: 19/20.
- *Ultimate:* Execute Step 1 and Step 2 by the checkpoint and Step 3 and Step 4 by the final delivery date. Max grade: 20/20.

## 10 Relevant Dates

- October $4^{th}$ - Electronic submission of the checkpoint code;
- October $7^{th}$ to October $11^{th}$ - Checkpoint evaluation during the lab sessions;
- October $25^{th}$ - Electronic submission of the final code and report.

## 11 Grading

The project grading will depend on a discussion at the end of the semester where all members of the groups must be present and where individual grades will be determined. That grade will depend on, besides the quality of the project, the individual performance in the discussion and the lecturer's evaluation.

The project grade (45% of the course's grade) is the *best* of the following two:

- Final_Project_Grade
- 85% of the Final_Project_Grade + 15% of Checkpoint_Grade

## 12 Cooperation among Groups

Students must not, *in any case*, see the code of other groups or provide their code to other groups. If copies of code are detected, all groups involved will fail the course.

## 13 “Época especial”

Students being evaluated during the “Época especial” will be required to do a different project and an exam. The “Época especial” project will be announced on the first day of the “Época especial” period, must be delivered on the day before last of that period, and will be discussed on the last day.