UNIVERSITY OF TARTU
Institute of Computer Science

Distributed Systems (Spring 2021)

Task 3: replication, consistency and fault tolerance

Practical information:
Due date: Monday, May 17, 23:59 (FIRM DEADLINE)
- This task can be performed in a team of max 3 persons ONLY.
- Total amount of pts that that can be collected = 100 + Bonus pts = 108 pts. **Bonus pts are optional**

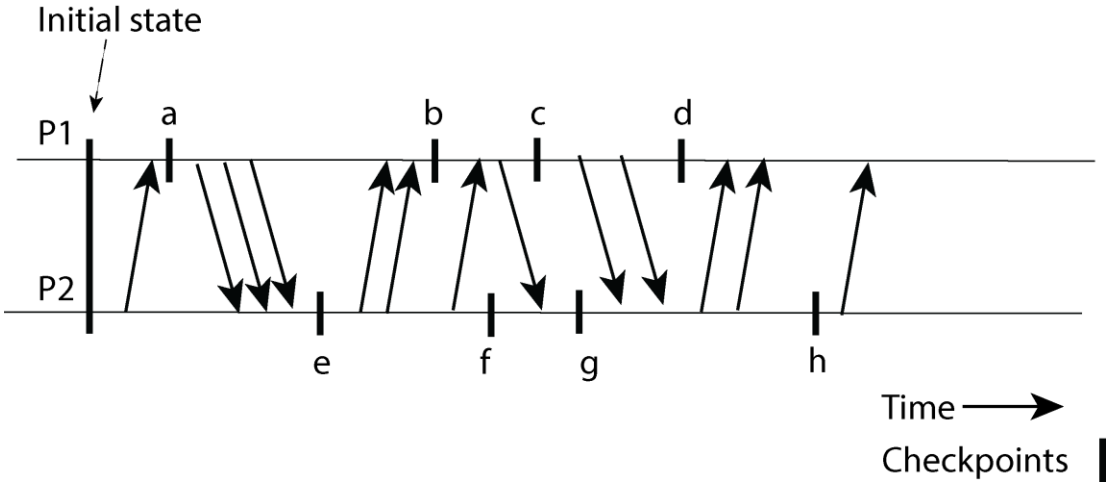
Submit your solution to farooq.ayoub.dar@ut.ee, mohan.liyanage@ut.ee, (CC) huber.flores@ut.ee

Instructions: Please respond to the following questions. **Be clear and concise in your answers.** Provide enough explanation to support your arguments. Ambiguous answers to fill up space are considered wrong and no points are granted. For questions that involve implementation, be sure to include everything needed to execute your program (Re-submissions are not allowed).

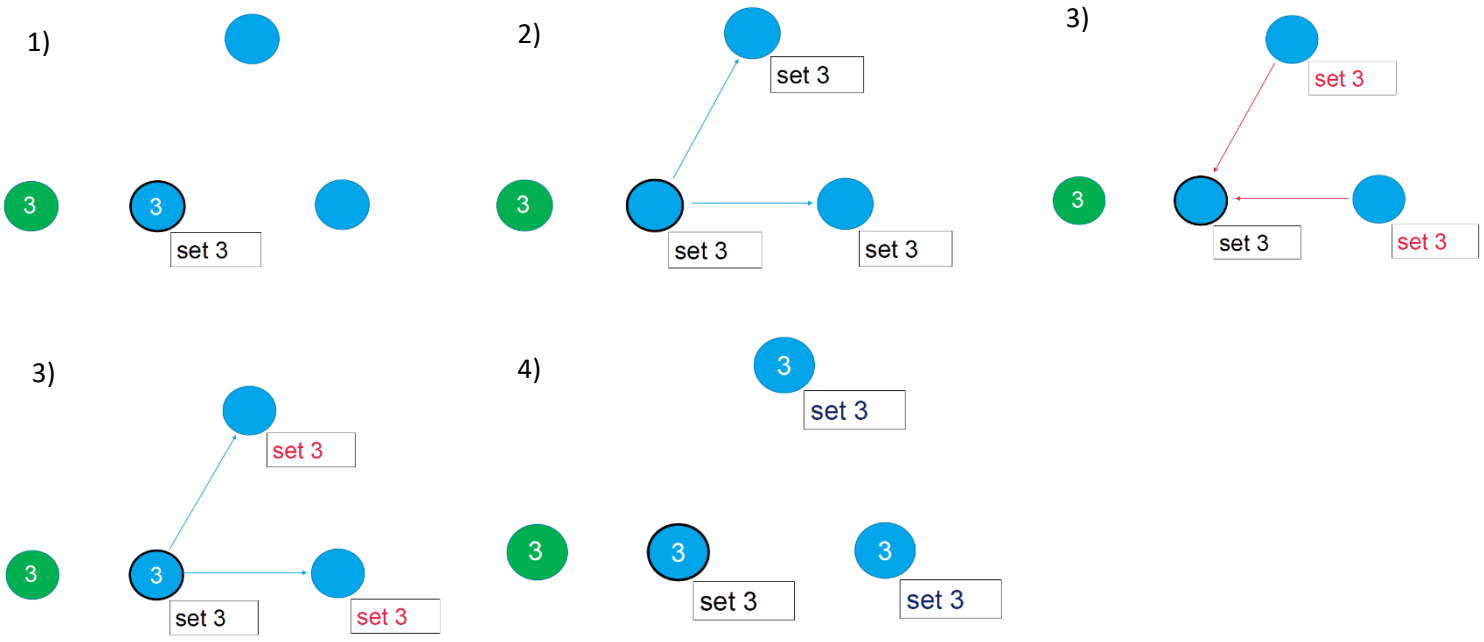
1. Elaborate based on consistency requirements.
   a. Describe the execution of these instructions, such that sequential consistency can be ensured (8pts).
   b. List all the cases in which causal consistency can be achieved (just removing instructions is allowed) (12pts)

P1:	W(x) A	R(y) C		W(x) C		
P2:	W(y) C		W(x) B	R(y) B		
P3:		R(y) C	R(x) A	R(x) B	R(x) C	
P4:		W(y) B	R(y) C		R(x) C	R(y) B
P5:		R(x) A		R(x) B	R(x) C	R(y) B

2. Consider the following figure below, where processes P1 and P2 are recording checkpoints independently (not synchronized). This means that there is a risk associated to achieve a global consistent checkpoint. Identify which checkpoint needs to remove from the timelines, such that the global consistency is not achieved, and instead, a domino effect is triggered. The domino effect implies that the last synchronization checkpoint between processes is the initial state. Describe the execution of instructions and provide detailed explanation of your reasoning. (10 pts)



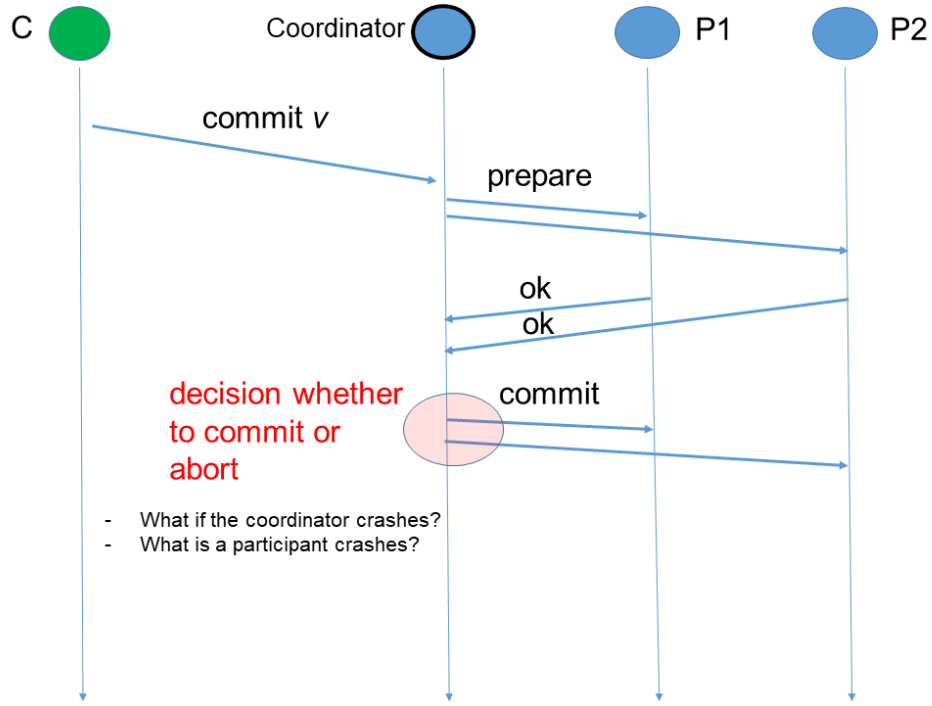
3. Assuming reliable communication in a system (meaning no messages are loss) that uses log replication, e.g., Raft.
    a. Derive a formula to model the number of messages that are exchanged between leader and followers (N nodes) to reach distributed commit consensus (10 pts) - the overall process is shown below with 3 nodes.
    b. Analyze how the number of messages increased as nodes are added in the system. Illustrate your reasoning with diagrams as required (5 pts).



4. Implement a simple two-phase commit protocol for a distributed system (2PC) - read slides 29-33 for more detailed information.

Fundamentals: while distributed commit (aka atomic commit) relies on fundamental consensus principles, e.g., voting, at the higher level, both consensus and distributed commit are different. In consensus, one or more nodes propose a value, and then any one of the proposed values is decided for the whole system. Moreover, crashed nodes can be tolerated as long as a quorum is working. In contrast, in distributed commit, every node votes whether to commit or abort a specific value. Nodes commit if all nodes commit, and must abort if >=1 nodes vote to abort. Moreover, if a participating node crashes, then the commit operation should be aborted.

2PC behavior – no failures: When assuming reliable communication (meaning no messages are loss), a process node "C" acting as client commits a value to the whole system of nodes through the "coordinator" node. Figure below shows the whole process without failures.



In the figure, we can observe the coordinator sending a pre-write message (prepare) to notify the systems about the upcoming operation. Next, every process (P1 and P2) send an acknowledgment to the coordinator (OK votes) and gets ready to receive the operation. The coordinator decides to commit or abort based on the collected votes.

Implementation requirements: Your program can be implemented in any language and you are allowed to use any programming primitives as long as the logic of the protocol is preserved. Your program must load its initial configuration from a file, which format is described as follow.

Filename: 2PC.txt

#System
P1
P2
P3
P4; Coordinator

#State
Consistency-history; 4,5,6,7,8,9

Where,

- **#System** defines the name of each process in the system and defines the coordinator. From the file format example, P1, P2 and P3 are participants, and P4 is coordinator.
- **#State** provides a consistency history of states, meaning the different values the system had committed as a whole from start to latest. For instance, first committed value is 4 and current (latest) value is 9.

$ ***two-phase-program*** 2PC.txt (starts the program) (5 pts)

After that, your program must provide a command line interface and implement the following commands. **Please print the history state of each node after every command.**

$ Set-value X (8 pts)
This command sends a new value "X" to the system (to be committed)

$ Rollback N (8 pts)
This command rolls back to previous values based on a number N of positions. For instance, following our previous example, "Rollback 2" should take the system back to value 7. For simplicity, all values after the rolled back value are forgotten by the system. In this example, all values after 7 are loss.

$ Add PX (7 pts)
It adds a new process participant to the system. Notice that the added process should be synchronized with the system, such that consistency is preserved. For instance, $ Add P7

$ Remove PX (7 pts)
It removes a process from the system. For instance, $ Remove P2

$ Time-failure PX time (10 pts)
It makes a specific process "PX" unavailable in the system for a period of "time" in seconds. This means that the process cannot be reached nor respond to any external request. After the time is over, the process becomes again available and continues its normal operations. For instance, $ Time-failure P2 120

$ Arbitrary-failure PX time (10 pts)
This commands flips any acknowledgement (Boolean value) of a specific process for a period of "time" in seconds. $ Arbitrary-failure P2 500, transform OK acknowledgments into CANCEL for the upcoming 500 seconds.

Handling errors: Notice that errors should be handled by providing a message warning through console. For instance, $ Rollback 25 -> "The system cannot reverse to that long state"

Deliverables for your 2PC program:

1- A short video (< 5 min) showing your program in execution (showing all the commands) and explaining your code.
2- Source code and everything needed for executing your program (binaries and input file). We will evaluate your work with multiple file configurations. Thus, do not hard code your solution just to work with the input file described here.

(8 pts) BONUS POINTS: One main disadvantage with 2-phase commit is that algorithm is blocked until coordinator is recovered (in the case coordinator has crashed). Extend your previous 2PC implementation to handle the issue, and explain in detail your proposed solution. Use illustrations and code snippets to support your ideas.