Chivit Chhoeun, Amirabbas Sherafatian, Kayahan Kaya
Distributed Systems
Spring 2021

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

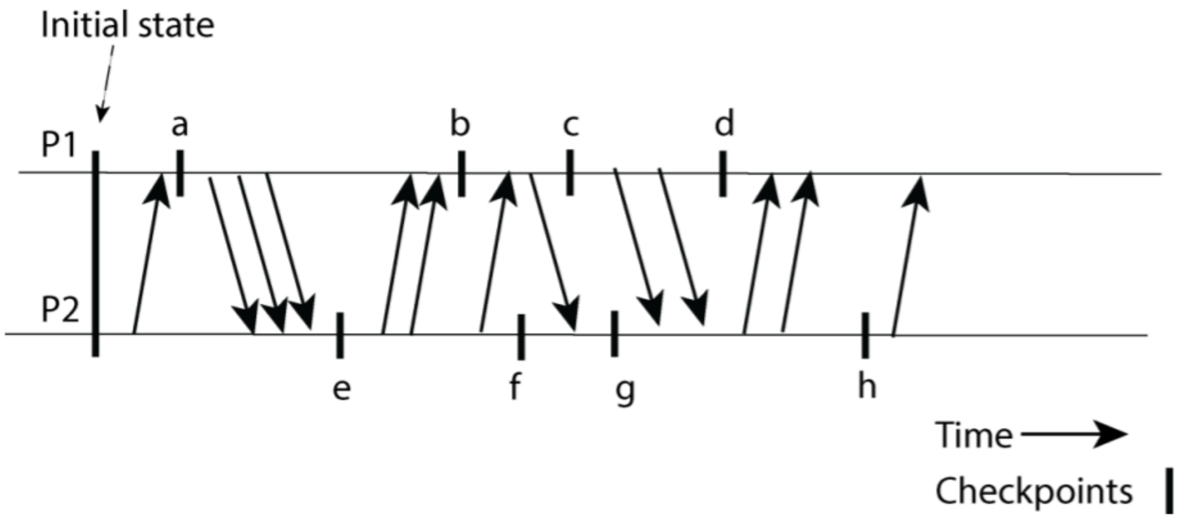
Answer:
1a. To achieve sequential consistency, the following instructions can be executed:

$P(2)$: $W(y)C$
$P(3)$: $R(y)C$
$P(1)$: $W(x)A$
$P(1)$: $R(y)C$
$P(5)$: $R(x)A$
$P(4)$: $R(y)C$
$P(3)$: $R(x)A$
$P(2)$: $W(x)B$
$P(2)$: $R(x)B$
$P(3)$: $W(x)B$
$P(4)$: $W(y)B$
$P(2)$: $R(y)B$
$P(1)$: $W(x)C$
$P(3)$: $R(x)C$
$P(4)$: $R(x)C$
$P(5)$: $R(x)C$
$P(4)$: $R(y)B$
$P(5)$: $R(y)B$

1b.
Here you can see all possibilities of operations which satisfy causal consistency.

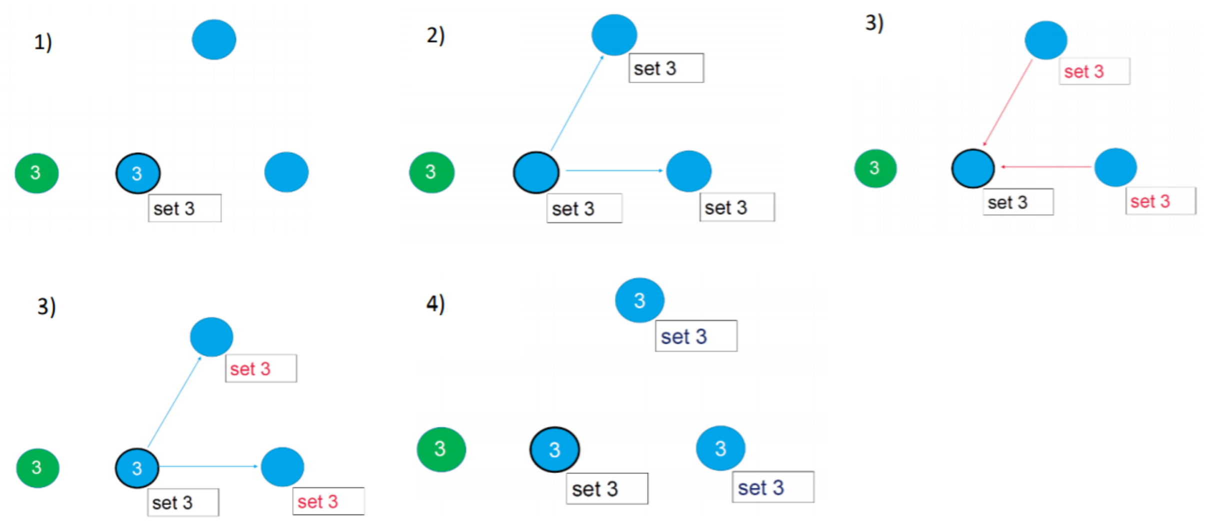
Order of operations	Results
$W_1(x)A, W_2(x)B, W_1(x)C , W_2(y)C, W_4(y)B$	$R_3(x)A, R_5(x)B, R_3(x)C, R_1(y)C, R_4(y)B$
$W_2(y)C, W_4(y)B, W_1(x)A, W_2(x)B, W_1(x)C$	$R_1(y)C, R_4(y)B, R_3(x)A, R_5(x)B, R_3(x)C$
$W_2(y)C, W_1(x)A, W_4(y)B, W_2(x)B, W_1(x)C$	$R_1(y)C, R_3(x)A, R_4(y)B, R_5(x)B, R_3(x)C$
$W_1(x)A, W_2(y)C, W_4(y)B, W_2(x)B, W_1(x)C$	$R_3(x)A, R_1(y)C, R_4(y)B, R_5(x)B, R_3(x)C$
$W_1(x)A, W_2(y)C, W_2(x)B, W_4(y)B, W_1(x)C$	$R_3(x)A, R_1(y)C, R_5(x)B, R_4(y)B, R_3(x)C$
$W_1(x)A, W_2(x)B, W_2(y)C, W_4(y)B, W_1(x)C$	$R_3(x)A, R_5(x)B, R_1(y)C, R_4(y)B, R_3(x)C$
$W_1(x)A, W_2(x)B, W_2(y)C, W_1(x)C, W_4(y)B$	$R_3(x)A, R_5(x)B, R_1(y)C, R_3(x)C, R_4(y)B$
$W_2(y)C, W_1(x)A, W_2(x)B, W_1(x)C, W_4(y)B$	$R_1(y)C, R_3(x)A, R_5(x)B, R_3(x)C, R_4(y)B$
$W_1(x)A, W_2(y)C, W_2(x)B, W_1(x)C, W_4(y)B$	$R_3(x)A, R_1(y)C, R_5(x)B, R_3(x)C, R_4(y)B$

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 be removed 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 a detailed explanation of your reasoning. (10 pts)



Answer:
Either the checkpoint c or g should be removed, then we will have a domino effect. As the figure shows, in the case of failure, if P1 rolls back to the check point c and P2 rolls back to the check point g, then we will have a globally consistent state (as it is known as distributed snapshot as well) from which the system can be recovered. With the current checkpoints, in the case of failure, if P2 rolls back to h and P1 rolls back to d, we don't have the global consistent state, since the checkpoint h has captured two messages sent to P1, but they are not captured in the checkpoint d by P1. So, P2 needs to roll back to g. In this case, P1 with the checkpoint d has captured two messages sent to P2, but the checkpoint g has not captured the receipt of these messages; Therefore, we have not reached a global consistent state. So this time, P1 again needs to roll back once more time to the checkpoint c. In this case, the checkpoint c in P1 has captured the last message sent to P2 and P2 with the checkpoint g has captured the receipt of this message, also there's not any messages sent to P1 from P2 which are captured by the checkpoint g and not captured by the checkpoint c. So, this means we have reached a global consistent state from which the system can be recovered from. If we didn't have either the checkpoint c or g, then these processes would need to roll back again and again until they would reach the Initial State so that we would have a cascaded rollback or domino effect.

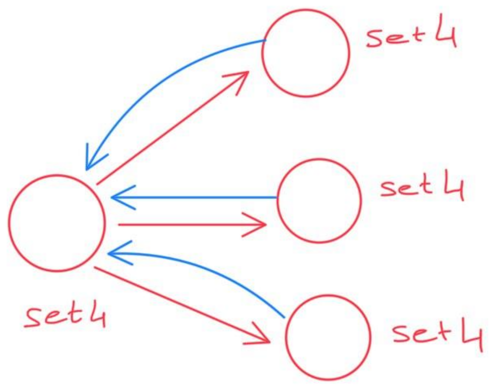
3. Assuming reliable communication in a system (meaning no messages are loss) that uses log replication, e.g., Raft.
    a. Derive a formula to model the number of messages that are exchanged between leaders and followers (N nodes) to reach distributed commit consensus (10 pts) - the overall process 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).



3a. Let Nbe the number of nodes. We can derive the number of total messages exchanged as:

$$Total\,messages\ =\ 3N\ -\ 3$$

3b. The number of messages increases as nodes are added because the candidate needs to send more messages/logs to all the followers, and vice versa. For example (see the image below), if there are 4 nodes, the total number of logs exchanged is 9. If there are 5 nodes, the total number of messages exchanged will be 12.



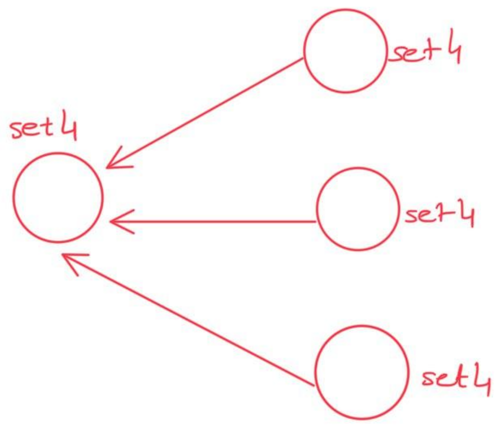
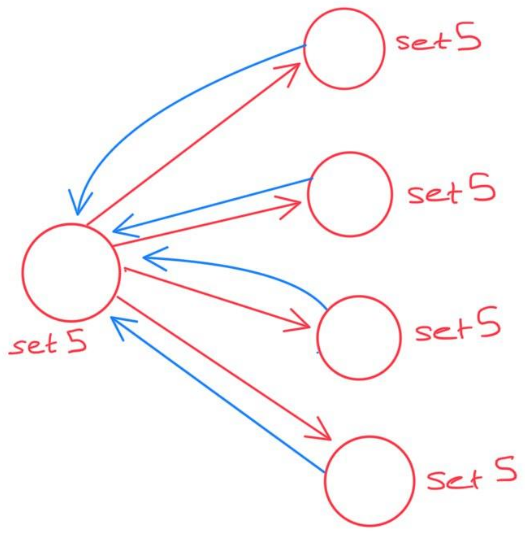


Figure 1 $\Longrightarrow$ logs = 9 $\Longleftarrow$ Figure 2



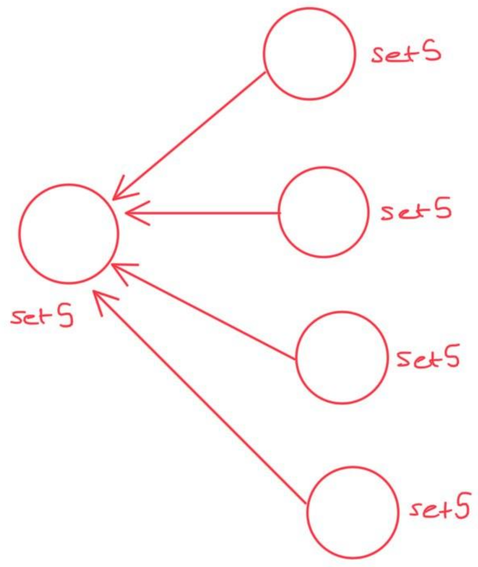


Figure 3 $\Longrightarrow$ logs = 12 $\Longleftarrow$ Figure 4

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.

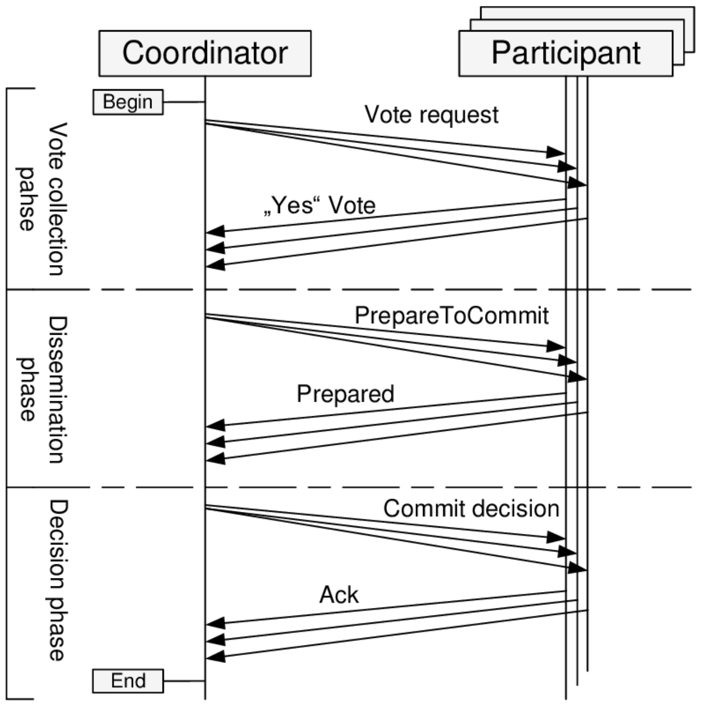
Answer:

In 2PC there's a main case in which the coordinator is crashed and all the processes need to be awaited until the coordinator is recovered, and this case is when the processes have responded with VOTE-COMMIT and are in the READY state. This is the main disadvantage of 2PC and because of this case, and the cases in which the coordinator and participants are blocked, 2PC is known as **blocking commit protocol**. To overcome this, an approach is to extend the 2PC to Three-phase commit (3PC).

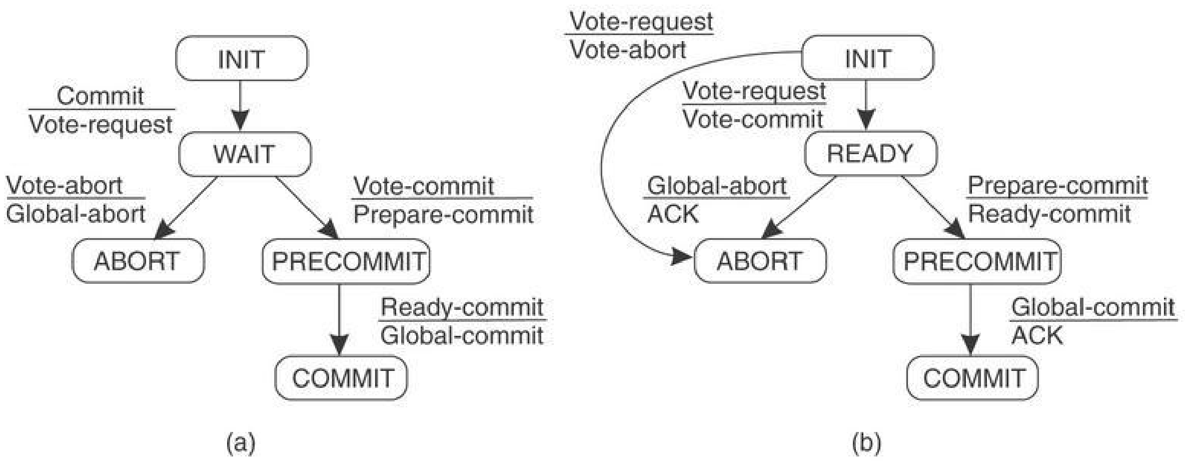
With 3PC we will go one step further while not all the blocking stages are covered while with fixing a timeout we can make the blocking time less. In the 3PC we have the same states as we had for the coordinator and participants, but another state is added which is PRECOMMIT. This state is applied to overcome the main case where some participants have got stuck in their READY state and are waiting for the coordinator to recover. Also in a case that the coordinator is blocked in its PRECOMMIT state and a timeout happens for some participants, the coordinator can handle the situation while it's in PRECOMMIT state.

With this approach, similar to 2PC at first the participants will go to INIT mode and send back to the coordinator a VOTE-COMMIT or VOTE-ABORT. If a VOTE-ABORT is received by the coordinator then it sends GLOBAL-ABORT to participants. If all the participants send VOTE-COMMIT then instead of sending a GLOBAL-COMMIT by the coordinator, the coordinator will send PRECOMMIT request to inform participants they need to be ready to commit and then after that, the coordinator will send the GLOBAL-COMMIT.

In the following figure, taken from www.researchgate.net the above mentioned procedure is shown.



Also, the states for the coordinator and participants are showing in the following figure taken from https://newbedev.com/how-does-three-phase-commit-avoid-blocking .


INIT
Commit / Vote-request
WAIT
Vote-abort / Global-abort
Vote-commit / Prepare-commit
ABORT
PRECOMMIT
Ready-commit / Global-commit
COMMIT
(a)
Vote-request / Vote-abort
INIT
Vote-request / Vote-commit
READY
Global-abort / ACK
Prepare-commit / Ready-commit
ABORT
PRECOMMIT
Global-commit / ACK
COMMIT
(b)

(a) coordinator, (b) participants

With the 3PC approach, similar to 2PC, if a participant (P1) gets stuck in INIT state, it can safely go to ABORT state when the timeout expires. Also, when it's in a READY state will reach another participant e.g. P2, if P2 is in ABORT state, P1 can go to ABORT state. If P2 is in COMMIT or PRECOMMIT state, P1 will go to COMMIT state. In addition, when the coordinator gets stuck in the PRECOMMIT and the timeout is expired, it can sends the GLOBAL-COMMIT, because the failed participant knows it has already sent the VOTE-COMMIT, so when it recovers, it can go to COMMIT by reaching other processes.

In the implementation, instead of sending GLOBAL-COMMIT right after all VOTE-COMMIT is received, a PRECOMIT is sent by calling PreCommit method in which we first send PRECOMMIT state to all participants and after that we call send the GLOBAL-COMMIT by calling Commit method :

```
def PreCommit(self):

    self.State = DSCoordinatorTransactionStatus.PRECOMMIT

    self.sendPreCommit()

    self.Commit()
```

```
def sendPreCommit(self):

    processes = dsProcessManager.GetParticipants()

    for p in processes:

        p.DSSocket.SendMessage(DSMessage(DSMessageType.PreCommit, self.Id))
```

```
def Commit(self):

    self.State = DSCoordinatorTransactionStatus.COMMIT

    self.sendGlobalCommit()

    self.onCommitHandler(self.Operation)
```

Also, in the case when a participant is blocked in its READY states and the timeout is expired, we should reach other participants to commit or abort. This part is done as follow:

```
def onReadyTimeout(self):

    # in the case of timeout when the participant is in the ready state, we should contact

    # other participants to find their state. we are deciding as follow based on the other prticipant's state:

    # if other participant's State is INIT => this participant will be ABORT

    # if other participant's State is ABORT => this participant will be ABORT

    # if other participant's State is PRECOMMIT => this participant will be COMMIT

    # if other participant's State is COMMIT => this participant will be COMMIT

       # if other participant's State is READY => we will reach another participant, if all others are in ready state,

    # we need to wait for the coordinator to be recovered

    if self.State == DSParticipantOperationStatus.READY:

        processes = dsProcessManager.GetParticipants()

        for p in processes:

            if p.Id != self.DSProcess.Id:

                            state = p.DSSocket.SendMessage(DSMessage(DSMessageType.GetParticipantState, self.CoordinatorTransactionId))
```

```
                if state == DSParticipantOperationStatus.INIT or state == DSParticipantOperationStatus.ABORT:

                    self.Abort()

                                        elif state == DSParticipantOperationStatus.COMMIT or state ==
DSParticipantOperationStatus.PRECOMMIT:

                    self.Commit()
```

And also, when the PRECOMMIT is sent by the coordinator and the participants receive it. The state in the participants is changed to PRECOMMIT and an acknowledge is sent back to the coordinator like below:

```
def PreCommit(self):
        self.State = DSParticipantOperationStatus.PRECOMMIT
        coordinatorProcess = dsProcessManager.GetCoordinator()

coordinatorProcess.DSSocket.SendMessage(DSMessage(DSMessageType.PreCommitA
cknowledge, self.CoordinatorTransactionId))
```