Distributed Computing
Project 2

Gaurav Nanda and Gurbinder Singh Gill

11/13/2013

In section 4.4, the paper (Paxos Made Moderately Complex) proposes an implementation of read-only commands, where the leader first runs a scout to determine whether its ballot is current, and only sends the read-only command to a replica after an adopted message is returned. However, this approach is not correct.

Problem: The above mentioned approach can allow a read operation to return a stale data (i.e. return a value that has been already overwritten).

Scenario: One possible execution to show this problem is as follows:

- ➔ Leader (L1) gets a Read-only (Op1) operation from one of the replicas.
- ➔ It runs a Scout (S1) and waits for an adopted message, thus making sure that its ballot number (B1) is current and no other leader has taken over and gotten a new command decided.
- ➔ Knowing that its ballot number (B1) is current, a leader sends Read-only commands to one of the replicas (R1).
- ➔ Since in an asynchronous environment message can be delayed. Lets say that replica R1 is running behind the overall system state and still have not received last few update commands, whereas rest of the replicas are up to date and ahead of this replica.
- ➔ Also there is no FIFO order, messages can reach replicas in any order, it is possible for ReadOnly (Op1) to reach replica R1 before update commands.
- ➔ If replica R1 replies at this stage, it is returning the value that has already been overwritten in the system, thus is a stale value.

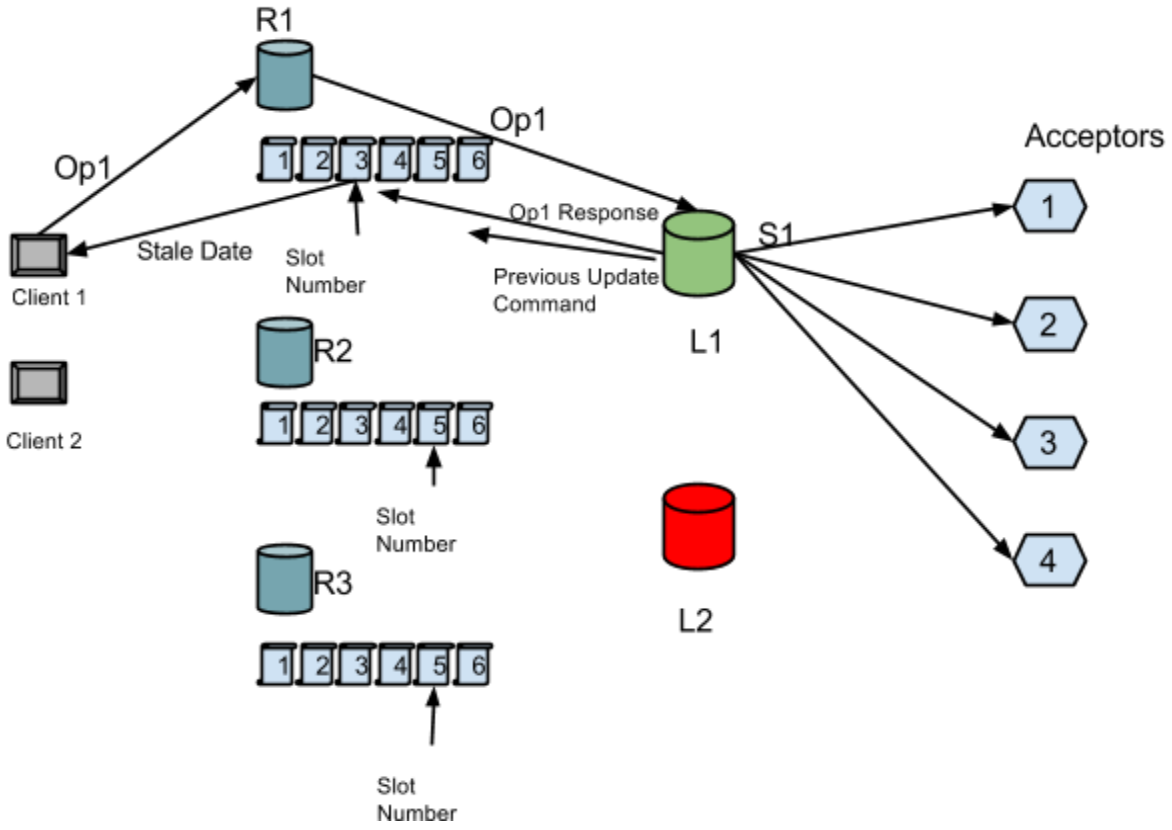

Fig1. Showing the above mentioned scenario.

To avoid the above mentioned scenario, the paper suggests to make replica R1 wait until any of the outstanding update commands have decided. One way to do this is to let the replica R1 know the highest slot number for which Leader L1 has proposed a proposal and got it accepted.

Not a solution: Even this can lead to stale value as described in the following scenario:

Scenario: Lets say that Leader L1 asks replica R1 to wait for Slot number SN before responding to the Read-only operation in an attempt to bring the whole system in one consistent state before response to ReadOnly is sent to the client.

- ➔ After letting Replica R1 know the slot number SN, Leader L2 (second leader) becomes active. This may be because L1 dies or stop responding to the pings of the Leader L2 (if failure detection is in place).
- ➔ Now Leader L2 becomes active and it gets ballot number adopted. Then it starts commanders for proposals in its proposal set.
- ➔ Leader L2 gets its proposal accepted for slot number higher than SN.
- ➔ The special Read-only message with SN sent by leader L1 is delayed and some of the replicas (except R1) move to slot numbers higher than SN as soon as they receive decisions for everything up to SN (except the outstanding Read-only command) and then decisions to the proposals from leader L2. But they do not know to send Read-only response on slot number SN.
- ➔ Replica R1 gets special Read-only command from leader L1 and is now waiting for a command for

slot number SN to come before it can respond to the Read-only request.

- ➔ As soon as R1 gets decision for slot number SN, it responds to the ReadOnly command, but it is now a stale value as system has moved to s slot number higher than SN.

Hence there is a problem with this solution.

Our Solution:

Implementation:

We are leveraging the idea of the solution given in the paper. As mentioned, whenever a read-only command comes to a leader, it sends a scout to get its ballot number adopted. However, doing only this much leads to the problems we mentioned above. To overcome this problem we are attaching special messages with update commands. Using those messages we try to make sure that if a leader has decided that any replica should execute that read-only command after slot number “X” and before “X+1”, then our protocol would ensure that no replica would execute an update command for a slot number greater than “X” before executing that read-only command.

Here is the overview of the changes we have done to the various entities:

Leader:

- ➔ Before executing a read-only command, leader attaches the “read-only command” with (its “maximum proposal” + 1) and sends it along with the scout to the acceptors.
- ➔ Leader doesn’t increase its ballot number when the read-only scout is spawned.
- ➔ Once it spawns such scout, it becomes **inactive** till it gets “Adopted” message back from the scout. This makes sure that new proposals received by this leader, are not executed until the scout comes back after updating the acceptors (We will discuss the acceptor updates in the following section).

Acceptors:

- ➔ Upon the receipt of this message from scout, acceptor checks if its adopted ballot number is equal to the ballot number sent in the request. If true, acceptor creates a dummy command message having (<slot-no(“max prop + 1”)> : read-only command) mapping and stores it in its pValues set.
- ➔ Going forward, when the leader (with the current ballot numbers) sends an actual update command for this slot, acceptor updates its PValue set such that it merges both the actual command and the dummy command.
- ➔ Why we do this/How it helps? (Discussed in the following section).

Replica:

- ➔ When commander would send an update command decided for a slot to the replica, replica would check whether that update command has a read-only(dummy) (or a list of read-only) command attached to this update command. If true, replica would go and execute that read-only command.

How this solves the problem:

- ➔ We are using the paxos memory (Acceptors) to store when a read-only command would be executed. This ensures us that in future, if any leader gets its ballot adopted, it would surely know when to execute that specific read-only command.
- ➔ This approach provides us the guarantee that read-only command and update commands would be

ordered in the same fashion across all the replicas, hence assuring us the safety.

- ➔ Re-iterating the previous points, even if one leader dies, it is assured that the new leader will definitely learn about the read-only commands sent by the previous leader and the new leader after getting its value accepted, would also send dummy(read-only) commands attached for the same slot.