Assignment 5
Group 5: Qi Lin, Xiaoying Li, Ziying Yu
6/3/2022

Question 1

Let's consider two cases of the basic version of the 2PC protocol described above - the commit case and the abort case. And write the transaction as T.

- Commit case:

First, the coordinator sends a prepare message to cohorts which respond with either a "yes" or a "no" vote. It's the local prepare work of the coordinator and cohorts. In this case, suppose cohorts vote "yes", and they write <Prepare T> on log records before sending prepared messages, which are forced writes to disk. And the coordinator also writes <Prepare T> on log records.

On receipt of cohorts' votes, the coordinator makes a decision to commit or abort the transaction. and suppose it decides to commit the transaction in this case. The coordinator writes <Commit T> on log records before sending the commit message, which is forced writes to disk.

Then the coordinator transmits the decision to the cohorts. Cohorts on receiving the decision, do the local commit work and send an ack to the coordinator. In this case, cohorts write <Complete T> on log records before sending acks, which are effectively forced writes to disk, though not immediately forced writes to disk.

On receiving all the ack, the coordinator can safely forget about the transaction and write <Complete T> on log records before deleting the entry of the transaction from its protocol database.

- Abort case:

First, the coordinator sends a prepare message to cohorts which respond with either a "yes" or a "no" vote. It's the local prepare work of the coordinator and cohorts. In this case, cohorts write prepare on logs (forced) if voting "yes", and cohorts voting "no" write lazy abort log records.

On receipt of cohorts' votes, the coordinator makes a decision to commit or abort the transaction. and suppose it decides to abort the transaction in this case. The coordinator writes <Abort T> on log records, which are forced writes to disk.

Then the coordinator transmits the decision to the cohorts. Cohorts on receiving the decision, do the local abort work and write <Abort T> on log records, which are forced writes to disk, though not immediately forced writes to disk. When done, send an ack to the coordinator.

On receiving all the ack, the coordinator can safely forget about the transaction and write <Complete T> on log records.

Question 2

At various stages of protocol, transaction waits from messages at both coordinator and cohorts. If the message is not received, on timeout, timeout action is executed.

If the coordinator times out waiting for an ack from the cohort for the message containing its decision, the timeout action it will take is to forward the transaction to the recovery process that will periodically send a commit to the cohort. When the cohort will recover, and all cohorts send an ack, the coordinator writes a completion log record and deletes the entry from the protocol database.

Question 3

- Explanation of what 2PC blocking means:

2PC is formed of two phases, namely a voting phase and a decision phase. During the voting phase, the coordinator of a distributed system sends a prepare message to ask all cohorts in the transactions' execution to prepare to commit the transaction, whereas, during the decision phase, the coordinator either decides to commit the transaction if all the cohorts are prepared to commit or to abort if any cohort has decided to abort.

When the coordinator fails, and at least one cohort keeps waiting for the final decision of the coordinator, such a scene depicts the 2PC as blocking, this means that cohorts cannot terminate the transaction (neither commit the transaction nor abort it) pending the recovery of the coordinator. If the coordinator crashes between the voting phase and the decision phase, a transaction can hold system resources for an unbounded period. Blocking means a non-failed process may have to wait for an unbounded time waiting for another process to recover.

- Explanation of under what circumstances 2PC is blocking:

Consider there is a coordinator and four cohorts c1, c2, c3, and c4, among which three cohorts c2, c3, and c4 are in the waiting state, which is another way for the prepare state that cohorts already vote for the transaction and are waiting for a decision. But while they are waiting for a decision, the coordinator and cohort c1 both failed. And there are two scenarios that might happen.

The first scenario is that cohort c1 is also prepared, so the coordinator decides to commit and send the message of commit to cohort c1, so the coordinator and cohort c1 committed, and then they both crashed. So now from the other cohorts' view, although they don't crash, they also don't commit.

The second scenario is that cohort c1 responds with a "no" vote, and then the coordinator aborts, so the state of the transaction is aborted. But then the other cohorts also don't know what happened.

Since both the above two scenarios might be true from the perspective of cohorts c2, c3, and c4, they cannot distinguish between these two scenarios. Because they cannot hear back from the coordinator, cannot know which scenario happens, and cannot know if cohort c1 responds with a "yes" or "no" vote. So, they are stuck because the fate of the transaction is decided by that vote. And this blocking scenario is the circumstance that 2PC is blocking.

Question 4

There are two phases in the Paxos algorithm - the leader election phase and the replication phase. In the leader election phase, a proposer attempts to become a leader by getting positive votes from a majority of nodes. In the replication phase, a leader replicates the value it wants to propose to a majority of nodes.

1.

In the leader election phase, each proposer associates its value with a unique ballot number and sends the prepare message to all acceptors. An acceptor only responds to a prepare(A) if A is the highest ballot number it has ever seen. If the leader election phase is removed, a node can become a leader without going through the leader election phase. Then consider a scenario where node A and node B both want to propose a value, since the leader election phase is removed, nodes A and B both become the leader immediately and enter the replication phase, where they both send their own proposal values to other nodes, which is contradicted the consensus safety condition that only a single value is chosen.

2.

Consider a scenario with three nodes A, B, and C, among which nodes A and C both want to propose a value. But node A fails after sending the propose message with value va to node B, and node C tries to become a leader and propose its value vc. Under normal circumstances, node C uses node A's value va since it receives it from node B's prepared message. In this case, from nodes B and C's perspective, they don't know what happened to va. It's possible that before node A crashed, it received the accept message from node B, and it responded to the client that va is the chosen value. And in order to make sure that any chosen value is going to be equal to that potentially chosen value va. So if node C uses the chosen value va, it can ensure that in the case where node A receives the accept message from node B and committed its own value va, the same value will be committed by node C, which is the most important property that there is only one chosen value. But here, suppose node C uses its own proposed value vc even if it received the previously accepted value va in the prepared messages from node B in the leader election phase, then if the above situation happens, the value va node A responded to the client may not be replicated by the same value, and the property of only one chosen value cannot be ensured.

3.

Consider a scenario with six nodes A, B, C, D, E, and F, among which nodes A and B both want to propose a value. Since in the leader election phase, instead of a majority, nodes A and B only collect 2 votes regardless of the total number of nodes. Suppose node A receives 2 votes from nodes C and D, node B receives 2 votes from nodes E and F, then they both become the leader immediately and enter the replication phase sequentially, where node A sends its own proposed value va to nodes C and D and receives accept message from nodes C and D, node B also sends its own proposed value vb to nodes E and F and receives accept message from nodes E and F. In this case, nodes A and B will both respond their own value to the client, which contradicted to the property that only one value can be chosen.

4.

Consider a scenario with six nodes A, B, C, D, E, and F, among which nodes A and B both want to propose a value, and node B has a higher ballot number than node A. Under normal circumstances, an acceptor only responds to a propose message if its ballot number is the highest ballot number it has ever seen, so acceptors will only respond to node B's propose message, and only node B can go through the leader election phase and become the leader. But here, a node responds to a propose message even if its ballot number is lower than the highest ballot number it has ever seen. Suppose nodes C and D respond to node A's propose message even if node A's ballot number is lower than node B, and nodes E and F respond to node B's propose message. In this case, the system cannot decide which one should be the leader, which leads to a conflicting leader situation.

5.

Consider a scenario where node A with ballot number b1 becomes a leader by getting positive votes from a majority of nodes in the leader election phase. But in the replication phase, node A uses a different ballot number b2 to propose a value. Suppose b2 is lower than b1. Since an acceptor only accepts a proposal with the highest ballot number it has ever seen, the propose message with ballot number b2 won't be accepted even if it's sent by node A, and the replication phase will fail to agree with the value proposed by node A.

6.

Under normal circumstances, Paxos uses unique ballot numbers to distinguish between proposed values. But here, two nodes may use the same ballot numbers. Consider a scenario where node A and node B with the same ballot number b both want to propose a value, and they both become leaders by getting positive votes from a majority of nodes. Then in the replication phase, node A proposes (b, va), node B proposes (b, vb), and some nodes may accept va and some may accept vb. Then a potential new leader node C will ask all nodes for their highest accepted value. It will then get back responses that are different values for the same ballot number. It cannot then disambiguate and decide which value is the one to choose next to keep consistency.

Reference:
https://stackoverflow.com/questions/45610160/why-does-paxos-proposalid-need-to-be-unique

Question 5

1.

Yes, the atomicity of transactions can be preserved. A protocol that guarantees atomicity can be designed as follows:

- Assume DBMS2(2) is the primary node, denote DBMS3 as 3, and DBMS1 as 1.
- 2 BEGIN and sent PREPARE to 3
- If 3 replies 'yes' to 2, 2 then send COMMIT to 1. If 1 successfully acknowledges 2 and 1 COMMIT, then 2 send COMMIT to 3 and 3 just COMMIT directly. If 1 does not acknowledge 2 and 1 ABORT, 2 then sends ABORT to 3 and 3 just ABORT.
- If 3 replies 'no' to 2 and 3 ABORT. 2 then send ABORT 1 and 1 ABORT after hearing from 2.

Thus, the atomicity of transactions can be preserved by making sure all DBMSs commit/abort.

2.

No, the atomicity of transactions cannot be preserved.

Since both 1 and 2 do not support a PREPARE primitive, they can do their own jobs directly without informing each other. Let's say 1 is ABORT and 2 is COMMIT, and 3 can be ABORT/COMMIT according to 1 or 2. In this case, the atomicity of the transaction cannot be preserved since a consensus cannot be formed in all DBMSs.

Question 6

A.

Transaction T's coordinator cannot receive the prepared message from the subordinate site, so it will abort T.

Since the subordinate fails before receiving a prepare message, it would not commit anything. When it covers, it will ignore T.

B.

Assume "making a decision" means force-writing a prepare/abort log.

The subordinate site first contacts the coordinator and sees how the coordinator will deal with it. If the coordinator decides to abort/commit, it sends an abort/commit message. When the subordinate site receives the abort/commit message and abort/commit transaction T.

C.

The coordinator first waits for the subordinate to respond. If no response is received after the timeout, the coordinate force-writes an abort commit log, then sends the abort message to T's subordinates.
When the subordinate recovers before timeout, it finds the abort log, so it sends no to the coordinator. After the coordinator receives no from that subordinate, the coordinate force-writes an abort commit log, then sends the abort message to T's subordinates.

D.

The coordinator first waits for the subordinate to respond. If no response is received after the timeout, the coordinate force-writes an abort commit log, then sends the abort message to T's subordinates.
When the subordinate recovers before the timeout, it finds the prepare log, so it sends prepared to the coordinator. After the coordinator receive prepared from that subordinate, the coordinate force writes a commit/abort log, then sends the commit/abort message to T's subordinates.

E.

In this case, the coordinator forces-writing an abort log record and sends an abort message to all the subordinates.
After the subordinate recovers, it asks about coordinator T's status. Since now the coordinator has already forgotten T, it replies abort. Then this subordinate will log abort T and forget T.

F.

In this case, the transaction does not happen. A transaction T is only decided after the coordinator site recovers from failure.

G.

In this case, the subordinates need to wait until the coordinator site recovers. Once the coordinator site recovers, the subordinate can then call the coordinator site. And this time, the coordinator site does not have any information regarding the commit/abort messages, so it can still abort T.

H.

In this case, the subordinates need to wait until the coordinator site recovers. Once the coordinator site recovers, the subordinate can then call the coordinator site.
The coordinator can abort the transaction because it finds the abort log record and no info about T, so it replies abort to the subordinate.

I.

It is the same scenario as the previous one.
The subordinates first wait for a period of time. After the coordinator recovers, the subordinate then contact the coordinator.
The coordinator can commit the transaction because it finds the commit log record and sees info about T, so it replies commit to the subordinate.

J.

For a commit case, it is not possible, since we know that the coordinator has received all the ack messages from the subordinates before it writes an end log record. So, the subordinates already know the status of T.

For an abort case, it is possible that we need to get T's status for some subordinates since they might not receive an abort message from the coordinator. The coordinates do not need to wait for all ack messages (actually there is no ack message) and directly write the end log under the presumed abort situation, so that's why some subordinates might not receive an abort message and need to ask coordinate about T's status during the recovery process.

Question 7

1.

A problem with the original Paxos protocol's condition is that all quorums intersect by simply requiring that quorums from distinct phases overlap in this work. In addition, Paxos' tradeoffs are rather tight, requiring a strict majority of hosts to be operational in order to enable writes.

Flexible Paxos, on the other hand, allows the designer more freedom in terms of constraint selection. It is not necessary to have a majority of nodes pledge and accept. Instead, as long as the quorums intersect, Flexible Paxos can employ any quorum.

2.

Paxos' safety and viability are guaranteed by the fact that any two quorums will intersect. The correctness of the disjoint and majority quorums in Paxos is the insight they observed for their solution. They also observed that phase 2 creates more overheads than in phase 1 in Paxos, which also gives them insights to solving the issue.

3.

Unlike the original Paxos, Flexible Paxos can eliminate the inherent tradeoff between resilience and performance by reducing the quorum intersection condition. And the new thing is that they reduce the original protocol's requirement that all quorums intersect to merely requiring that quorums from distinct phases intersect. FPaxos is strictly more general than Paxos. And the interesting thing is that FPaxos with intersecting quorums is the same as Paxos. One interesting thing is that the authors also observe that only phase 1 quorums need to intersect with phase 2 quorums, instead of requiring Q1's to intersect with each other and the same as Q2's quorums.

4.

Their approach is any group of at least N/2 + 1 acceptors can form a Q1 or Q2 quorum in FPaxos, so that Q2's size can be reduced to N/2 and keep Q1 the same. The term referring to FPaxos's quorum system is called simple quorums. In such systems, any acceptor can join a quorum, and each acceptor's vote counts equally. FPaxos requires |Q1| + |Q2| > N, which means that only quorums from different phases intersect. Furthermore, the leader in the system only sends prepare and propose messages to at least |Q1| or |Q2| acceptors. If any of them do not reply, the leader sends messages to more acceptors. An alternative to the simple quorums system is the grid quorums. It reduces quorums to one row plus any or one grid from each row below, and the average size becomes N1+(½)N2. Any single acceptor can form a Q2 but every acceptor must in Q1, which allows all acceptors to learn the decided value in one hop.

5.

The change of Q2's size reduces latency and improves the throughput and fault tolerance of the system. The simple quorum approach minimizes latency because leaders won't have to wait for a majority of participants to approve suggestions.

However, latency can also increase when the leader does not choose the fastest acceptors and must retransmit in the face of failures. A downside of the grid quorums system is that recovery from leader failure needs to wait for every acceptor to be up.

6.

A strength I like is that the authors introduce different techniques such as simple quorums and grid quorums to implement FPaxos. So, readers get the favor of multiple ideas, and they can pick one they like. And the inspiration from using Paxos' correctness to create another system that makes advancements is just spectacular.

Even though the authors thoroughly introduce three quorum systems for FPaxos, majority/simple/grid quorums. I wish they could talk about the whole processes in FPaxos with the usage of each system, like how they describe each of the stages in the original Paxos in detail. So that it can give me a more complete understanding of FPaxos implementation and its advancements.