15-440/15-640: Homework 2 Solutions

Due: October 12, 2018 11:59pm

Name:
Andrew ID:

1 Concurrency Control (18 points)

1. What is the major drawback of 2 phase commit protocol and explain in no more than three sentences how the 3 phase commit protocol overcomes it **[6 pts]**.

   The Two-Phase Commit Protocol is a blocking protocol. It can go to a blocking state by the failure of the coordinator when the participants are in uncertain state. The participants keep locks on resources until they receive the next message from the coordinator after its recovery. The 3PC protocol eliminates the 2PC protocol's system blocking problem with the third phase preCommit. If the coordinator fails before sending a preCommit message, other processes will unanimously agree that the operation was aborted. The coordinator will not send out doCommit message until all processes have acknowledged.

2. How will the 3 phase commit protocol ensure correct recovery when a coordinator crashes and a recovery node takes over the transaction? Assume that there is no write ahead logging here (otherwise recovery is trivial). **[6 pts]**

   If the coordinator crashes at any point, a recovery node can take over the transaction and query the state from any remaining nodes. If a node that has committed the transaction has crashed, we know that every other node has received a 'prepare to commit' message (otherwise the co-ordinator wouldn't have moved to the commit phase), and therefore the recovery node will be able to determine that the transaction was able to be committed, and safely shepherd the protocol to its conclusion. If any node reports to the recovery node that it has not received 'prepare to commit', the recovery node will know that the transaction has not been committed at any node, and will therefore be able to pessimistically abort.

3. How does 3 phase commit protocol handle a network partition failure? **[6 pts]**.

   Depending on how you consider the behavior of the nodes when they are in the 'prepare to commit' phase and did not hear back from the coordinator, you can give either answer. Any of the answer will be given credit if your explanation is valid.
   1. [Cannot handle] 3-phase commit protocol cannot handle the case of network partition. Imagine that all the nodes that received 'prepare to commit' are on one side of the partition, and those that did not are on the other. Then both partitions will continue with recovery nodes that respectively commit or abort the transaction, and when the network merges the system will have an inconsistent state.
   2. [Can handle] 3-phase commit protocol can handle the case of network partition. This is because only the nodes that are in 'prepare to commit' and can form a majority will continue to commit the transaction. Since the protocol proceeds on only on the side of the partition that contains the majority of the nodes, the system will be in a consistent state.

2 Distributed Mutual Exclusion (25 points)

Part A (8 points)

There are 6 processes with IDs from 0 to 5. The process with ID 5 is the leader, but after some time, it crashes. Process 2 notices that the leader failed.

To elect a new leader, these processes follow the Bully Algorithm (as mentioned in class) to elect a new leader. Process 2 notices that the leader failed.

1. Process 2 starts an election. Which process(es) does 2 send an ELECTION message to? **[2 pts]**

Processes with ID greater than it - 3 and 4 (optionally 5, even though it already noticed it failed).

2. List what processes send what messages (ELECTION, STOP, OK) until the group of processes come to consensus about electing a new leader **[8 pts]**.

Note: STOP and OK are equivalent in this context. (optional, implied from previous part) Process 2 sends ELECTION to 3, 4 (and optionally 5). Process 3 and 4 both send STOP to 2. Process 3 sends ELECTION to 4 (and optionally 5 if it did not detect that 5 failed). Process 4 sends STOP to 3. (optionally 4 sends ELECTION to 5 if it did not detect that 5 failed). Process 4 becomes the new leader.

Part B (17 points)

Consider three processes. The system has totally ordered clocks by breaking ties by process ID. It uses the Ricard & Agrawala algorithm. The timestamp for each process of id i is $T(p) = 10 * L(p) + i$, where $L(p)$ is a regular Lamport clock.

Each message takes 2 'real-time' steps to get delivered. Critical section takes 2 real-time steps. Fill in the table with the messages that are being broadcast, sent, or received between the processes until all nodes have executed their critical sections. Write 'execute critical section' as the action for a node when it enters its critical section. The first three rows have been filled in for you, and the fourth row has been started. Assume that if a process receives messages from the other two processes at the same time, the message that comes from the lower process ID will be received first.

Consider three processes. The system has totally ordered clocks by breaking ties by process ID. It uses the Ricard & Agrawala algorithm. The timestamp for each process of id i is $T(p) = 10 * L(p) + i$, where $L(p)$ is a regular Lamport clock.

Each message takes 2 "real-time" steps to get delivered. Critical section takes 2 real-time steps. Fill in the table with the messages that are being broadcast, sent, or received between the processes until all nodes have executed their critical sections. Write "execute critical section" as the action for a node when it enters its critical section. The first three rows have been filled in for you, and the fourth row has been started. Assume that if a process receives messages from the other two processes at the same time, the message that comes from the lower process ID will be received first.

Action Types: Broadcast (B), Receive (R), Send (S), Execute Critical Section (ExCS) Initial timestamps: P1 $\rightarrow$ 121, P2 $\rightarrow$ 242 and P3 $\rightarrow$ 103

Real Time	Process	Lamport Time	Action(to/from)	Contents	Q at P1	Q at P2	Q at P3
1	1	121	B	(request 121)	121		103
	3	103	B	(request 103)			
2	2	242	B	(request 242)	121	242	103
3	1	141	R from 3	(request 103)	103	103	103
	2	252	R from 1	(request 121)	121	121	121
	2	262	R from 3	(request 103)		242	
	3	133	R from 1	(request 121)			
4	1	251	R from 2	(request 242)	121	242	103
	1	261	S to 3	(reply 103)	242		121
	2	272	S to 3	(reply 103)			242
	2	282	S to 1	(reply 121)			
	3	253	R from 2	(request 242)			
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							

SOLUTION

Real Time	Process	Lamport Time	Action(to/from)	Contents	Q at P1	Q at P2	Q at P3
1	1	121	B	(request 121)	121		103
	3	103	B	(request 103)			
2	2	242	B	(request 242)	121	242	103
3	1	131	R from 3	(request 103)	103	103	103
	2	252	R from 1	(request 121)	121	121	121
	2	262	R from 3	(request 103)		242	
	3	133	R from 1	(request 121)			
4	1	251	R from 2	(request 242)	121	242	103
	1	261	S to 3	(reply 103)	242		121
	2	272	S to 3	(reply 103)			242
	2	282	S to 1	(reply 121)			
	3	253	R from 2	(request 242)			
5					121	242	103
					242		121
							242
6	1	291	R from 2	(reply 121)	121	242	103
	3	273	R from 1	(reply 103)	242		121
	3	283	R from 2	(reply 103)			242
7	3	293	ExCS	(CS from request 103)	121	242	121
					242		242
8					121	242	121
					242		242
9	3	313	S to 1	(reply 121)	121	242	
	3	323	S to 2	(reply 242)	242		
10					121	242	
					242		
11	1	321	R from 3	(reply 121)	121	242	
	2	332	R from 3	(reply 242)	242		
12	1	331	ExCS	(CS request from 121)	242	242	
13					242	242	
14	1	351	S to 2	(reply 242)		242	
15						242	
16	2	362	R from 1	(reply 242)		242	
17	2	372	ExCS	(CS request from 242)			
18							
19							

3 Logging and Crash Recovery (12 points)

1. List one advantage and one disadvantage of checkpointing in a log based recovery system [**6 pts**].

   Advantages: Need not scan the entire log to recover. Once a checkpoint is reached while traversing the log in reverse, the system can use that to construct the dirty page table and transaction table at that point for recovery. Disadvantages: May need to lock the whole disk, which will result in poor performance during checkpointing.

2. Assume that we are following ARIES logging and recovery protocol. Before we are able to completely recover from a crash, another crash happened. Is it possible to recover from both the crashes? Explain [**6 pts**].

   For each record that is undone in the recovery phase, a compensation log record (CLR) is created. This ensures that changes that were already undone are not undone again in the case of a recovery phase crash or restart.

4 Distributed Replication/Paxos (25 points)

Part A (13 points)

1. Why do we need a prepare phase in the Paxos algorithm? [**3 pts**]

   The prepare phase helps to find out any chosen values in the system. It also helps in blocking older proposers that have not yet completed choosing their value.
   Consider a case where multiple servers send conflicting accepts. In that case, if a receiver chooses one of them and then crashes, there is no way to determine which one was accepted. This is because there isn't a single leader. The prepare phase determines the leader by getting a majority agreement. To read moree: http://people.csail.mit.edu/alinush/6.824-spring-2015/stumbled/paxos-explained-from-scratch.pdf

2. Suppose one of the servers received an ACCEPT for a particular instance of Paxos (remember, each 'instance' agrees on a value for a particular account event), but it never heard back about what the final outcome was. What steps should the server take to figure out whether agreement was reached and what the agreed-upon value was? Explain why your procedure is correct even if there are still active leaders executing this instance of Paxos [**5 pts**].

   The best solution: The server should send messages to all servers asking whether they accepted an ACCEPT value, and if so, what value. If a majority respond with the same value, then the server knows that agreement was achieved. The procedure is safe even with active leaders, because a majority is a majority – the leader MUST hear from at least one other node about the value that was agreed upon.

3. Five servers $S1$ to $S5$ are using Basic Paxos protocol. $S1$, $S2$, $S3$ have accepted proposal 5 with a value A. Once this has happened, can $S4$ or $S5$ accept a different value B? Provide a clear explanation [**5 pts**].

   Yes. If $S1$, $S2$, and $S3$ that have accepted $< 5, A >$, other servers could still accept B if it has a stale proposal number. For example, $S4$ could prepare 3 and discover no values. Then $S1$ could prepare 5 on just $S1$, $S2$, $S3$. Then $S1$ could complete accepts on just $S1$, $S2$, $S3$. And $S4$ can still complete accepts on $S4$ and $S5$ with $< 3, B >$.

Part B (12 points)

Three servers $S1$, $S2$, $S3$ are using the Basic Paxos protocol. A proposer executes the protocol with an initial proposal number n and a value v_1. But it crashes at some unknown point during or after the execution of the protocol. Can the proposer now restart and re-execute the protocol from the beginning with the same proposal number n and a different initial value of v_2? If yes, prove that this is safe. If no, given a scenario that shows the safety condition being violated. Your answer should be as clear and precise as possible.

The slides in the class present a stronger version of Paxos i.e. the acceptor only accepts the proposal if it is greater than that of any previous prepare requests.
What are the safety conditions for Paxos? From the 'Paxos Made Simple' paper in your readings: The safety requirements for consensus are: 1. Only a value that has been proposed may be chosen, 2. Only a single value is chosen, and 3. A process never learns that a value has been chosen unless it actually has been.

The answer is **Yes** according to the lecture slides.
For the first condition, by the definition of the Paxos protocol, the chosen value must be the proposed value or the acceptedValue of one of the nodes. The acceptedValue of a node can also only be the acceptedValue of a node or a proposed value. Initially, the nodes do not have an acceptedValue, so in the base case, the acceptedValues are set to a proposed value because that is the only other option. Thus, this means that the chosen value must also be a value that was proposed, so the first condition is met.
For the second condition, we want to show that if a value has been chosen, another value cannot be chosen. We will assume that the safety condition holds prior to the proposal (n, v_1) (prior to the crash), so we will only consider violations of the condition that can arise as a conflict between proposals (n, v_1) and (n, v_2).
If the initial value v_1 was chosen, then we know that a majority (2) of the nodes responded to the proposer's Accept message with an ACCEPT-OK message. This means that these nodes have a value of n for minProposal and a value of v_1 for acceptedValue. Then, when the proposer attempts to start over with the proposal (n, v_2), these two nodes will reject the Prepare(n) message because n is not greater than their value of minProposal (n). Thus, the proposer will not get a majority of Prepare-OK messages from the nodes, so it will restart the protocol with a larger numbered proposal. This new proposal will now receive all Prepare-OK messages, and a majority of them will have an acceptedValue of v_1, so the proposed value will now be v_1. Thus, if this proposal is accepted, the chosen value will again be v_1, so only one value will be chosen.
If the initial value v_1 was not chosen, v_2 is the only remaining value that can be chosen, so only one value will be chosen.
Thus, we have shown that in both cases, only a single value will be chosen so the second condition is met.
For the third condition, we want to show that a process never learns that a value has been chosen until it has actually been chosen. By the definition of the Paxos protocol, once the proposer has received a majority of Accept-OK messages, it chooses the proposed value and commits this value, informing all processes of the selection. Since this is the only situation in which the processes are informed of a chosen value, the third condition is met.
Thus, we have shown that all the conditions are met, so this is safe.

In the original paper, the proposer responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted. The answer is **No** if you follow the paper. Different proposals must have distinct proposal numbers.

Here are the examples of something bad that can happen using 3 servers:

$S1$ completes Prepare($n = 1$) with $S1$, $S2$.
$S1$ completes Accept($n = 1$, $v = v_1$) with $S1$.
$S1$ restarts.
$S1$ completes Prepare($n = 1$) with $S2$, $S3$ (and discovers no accepted proposals).
$S1$ completes Accept($n = 1$, $v = v_2$) with $S2$, $S3$.
$S1$ responds to the client that v_2 has been chosen.
$S2$ completes Prepare($n = 2$) with $S1$, $S2$ and gets back:
from $S1$: acceptedProposal=1, acceptedValue=v_1,
from $S2$: acceptedProposal=1, acceptedValue=v_2,
$S2$ chooses to use v_1 arbitrarily.
$S2$ completes Accept($n = 2$, $v = v_1$) with $S1$, $S2$, $S3$.
$S2$ responds to some client that v_1 was chosen.

A different problem that can occur involves a request from before the crash being delivered after the crash:
$S1$ completes Prepare($n = 1$) with $S1$, $S2$.
$S1$ completes Accept($n = 1$, $v = v_1$) with $S1$.
$S1$ sends Accept($n = 1$, $v = v_1$) to $S2$ and $S3$, but they don't receive it yet.
$S1$ restarts.
$S1$ completes Prepare($n = 1$) with $S2$, $S3$ (and discovers no accepted proposals).
$S1$ completes Accept($n = 1$, $v = v_2$) with $S2$, $S3$.
$S1$ responds to the client that v_2 has been chosen.
Now $S2$ and $S3$ receive the Accept($n = 1$, $v = v_1$) request and overwrite their acceptedValue to be v_1.
The state of the cluster is now that v_1 is chosen, even though a client has been told that v_2 was chosen.

5 Fault Tolerance and RAID (20 points)

You buy 20 used hard drives. Each individual hard drive has the following characteristics:

Capacity: 600 GB and MTTF: 1 year

Recalling fault tolerance from lecture, you decide to build a RAID-0 array with the disks.

1. What is the effective capacity and MTTF of the RAID-0 array? **[3 pts]**

   Capacity = 600 GB * 20 hard drives = 12000 GB or 12 TB
   MTTF = 1 year / 20 = 365 days / 20 = about 18.25 days

2. What happens if one individual hard drive fails in the RAID-0 arrangement? **[3 pts]**

   Failure of a single disk renders all data useless.

After a closer look at your lecture notes, you realize that a RAID-5 array might be better for storage and data durability (in addition to speed).

3. What is the effective capacity, MTTF, and MTTDL of the RAID-5 array? **[3 pts]**

   Capacity = 600 GB * 19 hard drives = 11400 GB or 11.4 TB
   MTTF for a single disk is 1/20 of a year
   MTTDL is 1/19 of a year after the failure of the first disk (students could say 1/20 + 1/19)

4. What happens if one individual hard drive fails in the RAID-5 arrangement? **[3 pts]**

   Failure of a single disk can be tolerated (data can be recovered).

5. List one advantage and one disadvantage of mirroring (as in RAID-1) in no more than two sentences or bullet points **[4 pts]**.

   Advantage: redundancy/replication of data (for fault tolerance of a single disk failure)
   Disadvantage: halves the amount of data storage, (also accept that it is not tolerant of two failures of the same 2 mirrored disks)

6. List one advantage and one disadvantage of having a single parity disk (with no striping of the parity disk) in no more than two sentences or bullet points **[4 pts]**.

   Advantage: can recover data from a failed disk by using the others (see degraded read/write), only takes one disk for redundancy of data
   Disadvantage: parity disk is the bottleneck (which leads to slower/worse performance)