Problem Set 4

15-440/15-640 Distributed Systems Spring 2016

REFERENCE SOLUTION

Assigned: Tuesday April 19, 2016

Due: Tuesday April 26, 2016 (by the start of class)

Submission procedure:

- Create a .pdf of your answers and upload to Autolab**.**
- If you handwrite your answers, scan into .pdf before uploading. Use a proper scanner, not your smartphone camera. If you answers are illegible, you will receive a zero grade.

Question 1 (20 points)

Your first job after graduation from CMU involves configuring a replicated key-value store for different customers, depending on their observed workload characteristics. Consider simple Gifford voting for read and write operations, with identical nodes that each have exactly one vote.

A. Client-1 is using 5 replicas and his workload is extremely read-intensive. What read quorum and write quorum would you select? Explain your answer.

We want V_r to be as small as possible.
$V_r = 1,\ V_w = V = 5$

B. Suppose Client-1's workload is 99.99% reads and 0.01% writes. On each operation, a node may fail with probability $p = 0.1$. You may assume that failures are completely independent (i.e., they are not correlated). For the read and write quorum values you chose for (A), what is the probability that an operation will fail?

$V_r = 1,\ V_w = 5$
The probability of a read operation failing is $p^{V-(Vr-1)} = 0.1^5 = 1e\text{-}5$
The probability of a write operation failing is P(write fail) = 1 - (0.9)^5 = 4.1e-1
The probability of an operation failing is 99.99% * 1e-5 + 0.01% * 4.1e-1 = 5.1e-5 (or 5.095e-5)

C. Client-2's workload is extremely write-intensive, and she is using 4 replicas. What read and write quorum would you choose? Explain your answer.

We want V_w to be as small as possible.
$V_w = (V/2) + 1 = 3,\ V_r = V - V_w + 1 = 2$

D. Suppose Client-2's workload is 1% reads and 99% writes. On each operation, a node may fail with probability $p = 0.1$. As in (B), you may assume that failures are

completely independent (i.e., they are not correlated). For the read and write quorum values you chose for (C), what is the probability that an operation will fail?

$V_w = 3, V_r = 2$

The probability of a read operation failing is P(at least three nodes fail) = $4 * 0.1^3 * 0.9 + 0.1^4 = 3.7e\text{-}3$

The probability of a write operation failing is P(at least two nodes fail) = $6 * 0.1^2 * 0.9^2 + 4 * 0.1^3 * 0.9 + 0.1^4 = 5.23e\text{-}2$

The probability of an operation failing is 1% * 3.7e-3 + 99% * 5.23e-2 = 5.18e-2

Question 2 (20 points)

Modern processors provide a special instruction called `compare-and-swap`. If you disassemble an x86-64 program that contains this instruction, you will likely see something like this: *(Note: refresh your memory of x86 assembly language from 15-213/15-513/18-213)*

```
MOV QWORD PTR RAX, [RBP - offset1] ; load expected value into
register RAX
MOV QWORD PTR RCX, [RBP - offset2] ; load new value into register
RCX
LOCK CMPXCHG [RBP - offset3], RCX  ; Compare and swap; Note that the
prefix
                                   "LOCK" ensures that the memory
bus is
                                   locked for atomic execution
```

The instruction compares the first operand (memory operand) with the value in register RAX , and only copies the value of the second operand to the first operand if the comparison returned “equal”.

Now assume that you are the chief engineer of Intel’s next generation CPU. As a cost cutting measure, you decide to greatly simplify the chip design by removing instructions that are complex to implement in hardware. If you remove the `compare-and-swap` instruction and all its equivalents from the instruction set, which are the failure recovery mechanism(s) most likely be affected? Choose from the following: a) intentions lists b) write-ahead logging; c) shadowing. Justify your choice.

c) Shadowing. Since shadowing requires atomic compare and swap of pointers. The pointer to page before modification is compared against the pointer to page after modification, and if comparison returns success, the new value of the pointer is copied. In this manner, we could serialize all update operations to a page without having to worry about race condition because the validation phase and write phase have been combined into one atomic operation.

Question 3 (20 points)

PNC bank in Pittsburgh has many VIP customers who do business in the oil and shale gas industry. As part of global multi-party deals that involve many countries, these VIP customers often request PNC bank to transfer money to their counterparts in Moscow, Russia. The international interbank funds transfer mechanism uses two-phase commit to

execute such multi-party deals. Over many decades of operation of this mechanism, no bank or customer has ever never lost any money in the funds transfer process.

With the downgrade of the Russian economy, Internet connectivity in Russia has become unreliable. Russia and the US are often partitioned for many hours. PNC's VIP customers are getting annoyed with how long their transfers take now. They ask PNC for a feature that lets them cancel a transfer after they have given approval, but before they have received confirmation of the transfer. As PNC's Chief Information Officer, you are responsible for all IT operations. You realize that the feature being requested would effectively allow a VIP customer to change her "yes" vote in a two-phase commit to "no". Such a change would require PNC bank to undo the tentative debit of her account that occurred with her earlier "yes" vote.

PNC bank places high value on keeping its VIP customers happy. As CIO, you are under pressure from Customer Relations. Would you recommend implementation of this feature? Can it result in PNC bank or anyone else losing money? If yes, give an example sequence of events that shows how loss can occur. If no, explain why.

If this cancel means the bank always undo the tentative debit supposing transaction is not committed, then it should not be recommended. For example, considering the following operation sequences:

Suppose coordinator is Moscow

1. A transfer transaction begins between PNC US on behalf of Alice and Russia of Bob
2. Both parties have voted yes
3. Network between US and Russia is partitioned
4. Alice is out of patience and decide to cancel the transaction, PNC US allows to undo the debit since it has not received commit yet.
5. Coordinator sends out the commit message to both Bob and Alice, and Bob's account is credited.
6. When the network heals later, coordinator contacts PNC for money. Either PNC will lose money or coordinator will (or someone else, or it goes to a war)

Alternatively, if we only support a BEST EFFORT cancel action. It is allowed as all cancel requests after the commit in coordinator happens would fail.

Question 4 (20 points)

You are running Paxos on a small cluster. Let "`I→J`" denote a successfully delivered message from node I to node J. Recall the messages used in the Paxos version discussed in class:

- `Prepare(n)`
- `Promise(n, already_accepted(n_k, a_k))`
- `PleaseAccept(n, a_k of highest n_k seen)`
- `Accept_OK()`

.

A. Assuming all nodes start out with (NULL, NULL) as their stable storage values, show the stable storage values resulting from the following scenario.

Participating nodes: A, B, C

```
A➞A: Prepare(42)
A➞B: Prepare(42)
A➞C: Prepare(42)
A➞A: Promise(42, null)
B➞A: Promise(42, null)
B➞C: Prepare(43)
C➞B: Promise(43, null)
B➞A: Prepare(43)
B➞B: Prepare(43)
```

43 is transacted.

B. Assuming all nodes start out with (NULL, NULL) as their stable storage values, show the stable storage values resulting from the following scenario.

Participating nodes: A, B, C, D, E

```
A➞A: Prepare(42)
A➞B: Prepare(42)
D➞D: Prepare(43)
D➞E: Prepare(43)
A➞A: Promise(42, null)
B➞A: Promise(42, null)
D➞D: Promise(43, null)
D➞E: Promise(43, null)
```

42 or 43 could be transacted, it depends on what prepare message(s) C gets

C. Your colleague is not happy with the performance of your production Paxos cluster and has decided to optimize the implementation by removing what he describes as an “unimportant conditional statement.” Upon receiving `PleaseAccept` messages, nodes no longer have a conditional and will always write `(n, highest_a_k)` to stable storage.

   a. Informally explain why this is not a valid optimization

      A prepare with a higher n would normally disrupt a node from accepting a lower-value commit. However with this check removed, a lower value commit can still succeed and multiple leaders be elected concurrently.

   b. Provide a message trace of a scenario where this modification causes an inconsistent system state.

      Nodes: A, B, C

      - A➡A: Prepare(42)

- A➡B: Prepare(42)
- A➡C: Prepare(42)
- A➡A: Promise(42, null)
- B➡A: Promise(42, null)
- C➡A: Promise(42, null)
- C➡C: Prepare(43)
- C➡B: Prepare(43)
- C➡A: Prepare(43)
- A➡A: PleaseAccept(42, v1)
- A➡B: PleaseAccept(42, v1)
- A➡C: PleaseAccept(42, v1)
- A➡A: Accept_OK()
- B➡A: Accept_OK()
- C➡A: Accept_OK()
- C➡A: PleaseAccept(43, v2)
- C➡B: PleaseAccept(43, v2)
- C➡C: PleaseAccept(43, v2)
- A➡C: Accept_OK()
- B➡C: Accept_OK()
- C➡C: Accept_OK()
- A and C are leaders.

Question 5 (20 points)

A. Consider a 10-disk array in which each disk has an MTBF of 100 years. If no redundancy is used, how soon do you expect data loss to occur?

We assume the time to failure follows a exponential distribution, and the failures are independent between disks.

The MTTDL = 100/10 years = 10 years.

B. Suppose redundancy is added to the array so that it can tolerate one crash, but not more. You may assume that each disk has a MTBF of 100 years and that no repair is possible. How soon do you expect data loss to occur ?

The expected time that first failure occurs is 10 years as in A. By memoryless property of exponential distribution, the expected time that the second failure occurs is 100/9 years = 11 years. Thus MTTDL = 21 years

C. Suppose 1000 disks are organized into 100 x 10-disk arrays, and each disk has an MTBF of 100 years. Once failed, disks are not repaired. Each array is organized as in B). Assuming that failures are independent across arrays, how soon can data loss be expected to occur?

A single array has MTTF of 21 years as in B. Thus the MTTDL for the 100 arrays is 21/100 years = 0.21 years