Problem Set 4

15-440/15-640 Distributed Systems Spring 2017

Assigned: Tuesday, April 25, 2017

Due: 10:30AM on Tuesday, May 2, 2017

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 your answers are illegible, you will receive a zero grade.

Question 1 (10 Points)

What type of replica control, *pessimistic* or *optimistic*, will you use for the following systems? Explain your answers very briefly.

A. A flight ticket booking system, implemented on top of a wide-area, replicated distributed database
B. A system for electronic stock market, implemented on top of a wide-area, replicated distributed file system
C. A news portal system (e.g., New York Times), using CDN
D. A multiplayer online game, where the players move figures around a common scene and the state of the game is replicated at the players' workstations and at a server

Solution:

A. **Pessimistic**, since we would like to guarantee strictly consistent information regarding seats, overbooking, etc.

B. **Pessimistic**. Sensitive financial information involved.

C. **Optimistic**. Temporary stale reads of news through CDN is generally tolerable.

D. **Pessimistic argument**: Need to ensure absolute consistency regarding game states.
Optimistic argument: Other factors may come into picture (latency, etc.), so the tradeoff makes pessimistic control less appealing. Could tolerate minor inconsistencies.

Both answers are accepted provided that a reasonable justification is included.

Question 2 (25 Points)

Typical transaction systems assume volatile memories and allow pages of uncommitted transactions to be evicted from memory to disk (e.g., using LRU or ARC). In addition, they do not usually flush pages of committed transactions to disk, mainly due to a bandwidth limitation in disks. However, non-volatile memories (NVMs) will soon become prevalent. These NVMs are persistent and offer greater bandwidth as compared to disks. Consider a system, referred to as ***NVSys***, with a modern disk and an NVM rather than a volatile memory.

A. Would using a logging sub-system with a Write-Ahead-Log (WAL) like protocol on top of ***NVSys*** necessitate redoing any type of transactions after a failure? If so, which transactions, committed or uncommitted, need to be redone and why?

B. Would using a logging sub-system with a WAL-like protocol on top of ***NVSys*** necessitate undoing any type of transactions after a failure? If so, which transactions, committed or uncommitted, need to be undone and why?

C. Would using a logging sub-system with a WAL-like protocol on top of a virtual machine with (hypothetically) unlimited volatile memory size necessitate redoing any type of transactions after a failure? If so, which transactions, committed or uncommitted, need to be redone and why?

D. Would using a logging sub-system with a WAL-like protocol on top of a virtual machine with (hypothetically) unlimited volatile memory size necessitate undoing any type of transactions after a failure? If so, which transactions, committed or uncommitted, need to be undone and why?

E. As we studied in class, distributed transaction protocols like 2PC make use of logging to achieve resiliency against specific failures. Can logging be entirely excluded from 2PC and still achieve such a resiliency if the volatile memory on each 2PC node is replaced with an NVM? Explain your answer.

Solution:

A. Nothing. The committed will be reflected in the NVM, so there is no need to redo it. (5 pts)

B. The uncommitted transactions have to be undone in order to ensure atomicity. Since upon failure recovery these uncommitted ops are still in the memory (which is non-volatile), they have to be removed as soon as possible. (5 pts)

C. Need to redo the committed transactions since given unlimited volatile memory the changes may not be flushed to the disk. This necessitates the redo on the successful commits. (5 pts)

D. No, everything is lost since we are using volatile memory here (and uncommitted changes shouldn't be kept anyway). (5 pts)

E. No, since logging is necessary to undo uncommitted transactions. This is necessary for the failure handling in 2PC.(5 pts)

Question 3 (20 Points)

Now that you are almost done with the distributed systems course at CMU, you started getting offers for developing real-world distributed systems. One offer came from MyBlogPost.com, a popular website for posting and following blogs, asking you to build a highly-available storage system for them. You accepted the offer and, after much thought, decided to use Gifford's quorum consensus replication as a technique to achieve high availability. You analyzed the user accessibility patterns at MyBlogPost.com over the past 12 months and realized that they can be classified into two major categories (say, CAT1 and CAT2). For efficiency reasons, you planned to treat the two categories differently. In addition, to keep your technique simple, you decided to give a weight of 1 to each voting node in the read and write quorums of each category.

A. Based on your analysis, CAT1 is read-intensive, with 98% read and 2% write accesses. You are planning to set up a 5-replica system for this category and seeking to optimize for the common case (i.e., make reads fast). What read and write quorum sizes would you select so as to achieve fast reads, while maintaining correctness? Explain your answer.

B. Based on your analysis, CAT2 is write-intensive, with 99% write and 1% read accesses. You are planning to set up a 4-replica system and seeking to optimize for the common case (i.e., make writes fast). What read and write quorum sizes would you select so as to achieve fast writes, while maintaining correctness? Explain your answer.

C. Suppose on each access, a node may fail with a probability, p = 0.1. You may also assume that failures are independent. For the read and write quorum values you computed in part B, what is the probability that an access will fail?

Solution:

A. (5 points) N = 5
R = 1 (read intensive)
By Gifford voting, R + W > N and W > N/2
=> 1 + W > 5
=> W = 5
Read Quorum: 1, Write Quorum: 5

B. (5 points) N = 4
By Gifford voting, R + W > N and W > N/2
=> W > 2
=> W = 3
R + W > N

=> R + 3 > 4
=> R = 2
Read Quorum: 2, Write Quorum: 3

C. (10 points) W = 3, 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-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-2
The probability of an operation failing is
= 1% * 3.7e-3+ 99% * 5.23e-2
= 5.18e-2

Question 4 (25 Points)

You have been nominated as the president and official web guru of DundeeAndMe.net, an unofficial fansite paying tribute to Paul Hogan and all things Crocodile Dundee. Due to the incredible growth of the site's popularity, you realized that having just a single web server can no longer handle the load. Hence, you decided to replicate the server, placing three replicas in different Australian cities (where the majority of the fan base resides), a single replica in southern Germany (where there is a small, but devoted cult), and a fifth replica in Boca Raton, Florida.

A. After having completed Project 4, you decided to use 2PC among the replicas for performing updates on site contents. List two advantages of using 2PC in this situation.

B. A freak tsunami destroyed the physical termination point of the main undersea cable connecting Australia and Asia. The Internet was effectively partitioned, separating Australia from the rest of the universe. Explain how this will affect users in Australia and in Germany, both when browsing the site as well as creating new blog entries.

C. Suppose you decided to use a Gifford-based voting scheme instead of 2PC to manage the replicas. In addition, assume a write quorum of 3 and an equal voting weight of 1 among all servers. Under the same partition conditions of part B, explain how users in Australia and in Germany will be affected, both when browsing the site as well as creating new blog entries.

D. Suppose you still wanted to employ Gifford voting with an equal weight of 1 across all servers, but now with a read quorum of 2. Again, under the same partition conditions of part B, explain how users in Australia and in Germany will be affected when browsing the site and creating new blog entries.

E. After some long daunting days, the undersea cable was finally restored, putting an end to this unpleasant network partition. During this time, however, the fortunate users who were able to continue using your site made several updates to the site's contents, which of course were not reflected on all replicas. Assuming a Gifford-based replica management scheme like the one discussed in part C, what recovery steps (if any) should you apply in order to ensure proper functioning of the replica set once network connectivity is restored?

Solution:

A. All replicas will stay in sync since 2PC requires unanimity. Browse requests can be sent to any replica, high read availability.

B. No further updates are possible, since 2PC cannot complete (requires server as well as all participants to vote "yes"). Browsing still works in Germany and in Australia, since replicas are available in each partition.

C. Write quorum of 3, so must have read quorum >=3. Assume read quorum=3. Can continue to browse and create blog entries in Australia. Cannot do either in Germany.

D. Read quorum of 3, so write quorum >= 4. Assume write quorum=4. Can continue to browse everywhere, but cannot create / update blogs anywhere.

E. No further recovery steps are needed. The read quorum should ensure that at least one of the replicas that are up-to-date are included, so all further reads will get latest data.

Question 5 (20 Points)

CNP bank uses a group of servers to coordinate and synchronize critical banking transactions. In particular, a Paxos-style protocol is employed on all the servers so as to achieve consensus on each operation. For any submitted operation, let $I \rightarrow J$ denote a successfully delivered message from server I to sever J. Recall the 4 types of messages that are used in Paxos, which can be written as follows:

- Prepare(n), where n is a unique sequence (or round) number
- Promise(n, (n_k, a_k)), where n_k and a_k are the last sequence number and value, respectively accepted by Acceptor k (if any) and stored at its stable storage
- PleaseAccept(n, v), where $v = a_k$ of highest n_k seen among the promise responses, or any value if no promise response contained a past accepted proposal
- Accept_OK()

A. Assume 3 servers, S1, S2, and S3 are involved, and they all start out with no past accepted proposals (i.e., no sequence numbers and values are stored at their stable storages). A server can act as a Proposer, an Acceptor, or both. Suppose also that servers S1 and S2 submit

proposals to quorums of Acceptors as shown in the Paxos communication trace below. What are the quorum sizes of S1 and S2, assuming that both will not submit prepare messages other than what is shown in the trace? Also, which proposal (or proposals) will achieve consensus? Explain your reasoning.

S1➔S1: Prepare(101)
S1➔S1: Promise(101, null)
S1➔S2: Prepare(101)
S1➔S3: Prepare(101)
S2➔S1: Promise(101, null)
S2➔S3: Prepare(102)
S3➔S2: Promise(102, null)
S2➔S1: Prepare(102)
...

B. Assume now 5 servers, S1, S2, S3, S4, and S5 are involved, and also starting out with no past accepted proposals. As in part A, a server can act as a Proposer, an Acceptor, or both. Suppose also that servers S1 and S4 submit proposals to quorums of Acceptors as shown in the Paxos communication trace below. Which proposal (or proposals) will achieve a consensus in this case, assuming the [EVENT!] in the trace is S1 crashing? Explain your reasoning.

S1➔S1: Prepare(101)
S1➔S1: Promise(101, null)
S1➔S2: Prepare(101)
S2➔S1: Promise(101, null)
S4➔S4: Prepare(104)
S4➔S5: Prepare(104)
S4➔S4: Promise(104, null)
S5➔S4: Promise(104, null)
S1➔S3: Prepare(101)
S3➔S1: Promise(101, null)
S1➔S1: PleaseAccept(101, X)
S1➔S1: Accept_OK()
S4➔S1: prepare(104)
S1➔S2: PleaseAccept(101, X)
S1➔S3: PleaseAccept(101, X)
[EVENT!]
…

Solution:

A. Since there are 3 participant nodes, the quorum size is 2. Only proposal 102 will be accepted (assuming no other proposals coming in), because at the end of the trace provided, both S1 and

S3 should have updated their value of Np to 102--- which means they will deny the subsequent PleaseAccept(...) on 101.

Credit may be given to students answering “either 101 or 102” if they **explicitly** assume that “S1➔S2” means successful send rather than delivery.

10 points
[5]: attempted
[2]: correct quorum size
[1]: 102 commits
[2]: good explanation (partial credit can be given)

B. S1 (proposal 101) will reach consensus first (b/c S2 and S3 shall reply Accept_OK()); but after it crashes, S4 (proposal 104) will eventually reach consensus as well (it has a quorum size of 3, and S2 & S3 will accept its proposal finally), but with the value proposed previously by S1!

Depending on how student understands “consensus”, the answer may not include proposal 101. But to receive full credit this must be stated clearly.

According to Piazza definition, consensus is the point where a proposal is positively accepted by a majority of acceptors in the quorum. So in this case, while S1 crashed, its proposed value is accepted by a simple majority.

10points
[5]: attempted
[2]: correct conclusion - proposal 104
[3]: explain why and what consensus is - value proposed by S1 (partial credit can be given)