Name: _____ Entry number: _____

I do hereby undertake that as a student at IIT Delhi:
(1) I will not give or receive aid in examinations, and
(2) I will do my share and take an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honour Code.

Q1 [4 marks] Virtualization

Q1.1 [1 mark] [True/False] When guest OS changes CR3, it must trap into hypervisor when we are using extended page tables for memory virtualization.

_____

Q1.2 [1 mark] [Tick correct answer] What is the Popek-Goldberg Theorem's requirement for building a trap-and-emulate hypervisor?

[ ] Privileged instructions ⊆ Sensitive instructions

[ ] Sensitive instructions = Privileged instructions

[ ] Sensitive instructions ⊆ Privileged instructions

[ ] Privileged instructions ≠ Sensitive instructions

Q1.3 [1 mark] [True/False] Original x86 architecture followed Popek-Goldberg Theorem.

_____

Q1.4 [1 mark] [True/False] QEMU on x86 runs in ring-0.

_____

Q2 [4 marks] Dynamo and Dotted version vectors

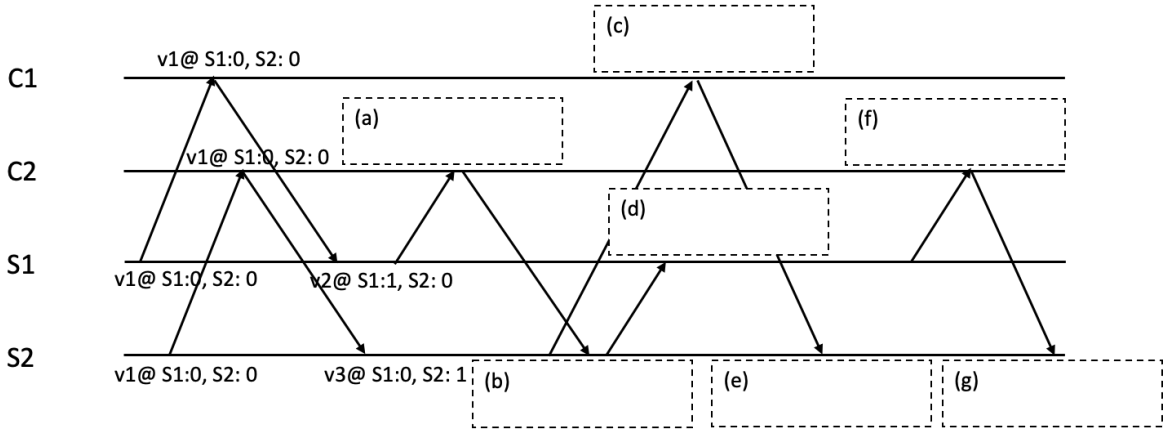
Q2.1 [1.5 mark] What is PACELC theorem?

Q2.2 [2.5 marks] Assume that S1 and S2 are two dynamo servers holding values for the key "K". For the following sequence of GET and PUT events on the same key "K", fill the value(s) and their timestamps.

Recall that each server may maintain multiple values of the key and that the GET may return those multiple values. Further, the next PUT is assumed to do "read repair" that

Name: _____ Entry number: _____

merges the read values. Assume that each PUT writes a new value like v4, v5 and so on. Assume that we are maintaining versions as dotted version vectors.



Q3 [11 marks] Spanner TrueTime

Google observed the worst case clock drift of 200 microseconds per second in their servers. Each server synchronises with time master server(s) every 30 seconds to keep TrueTime uncertainty in between 1 and 7 milliseconds.

Let us say that we change this behaviour to make every server synchronise its time with time master server(s) every 10 minutes instead.

Q3.1 [1 mark] What will be the maximum TrueTime uncertainty? Assume that all synchronisation attempts are successful, i.e, no network or server failure.

Let us now examine the impact of this change. For each of the following workload, mention if the workload is expected to become slower or will it remain unchanged. Justify your answer.

Q3.2 [2.5 marks] Snapshot read at the latest timestamp.

Name: _____ Entry number: _____

Q3.3 [2.5 marks] Snapshot read at a one day old timestamp (for backup and audit purposes).

Q3.4 [2.5 marks] Prepare phase of read-write transaction with no other conflicting transactions.

Q3.5 [2.5 marks] Prepare phase of read-write transaction with other conflicting transactions.

Name: _____ Entry number: _____

This page is intentionally left blank. You may use it for rough work.

Name: _____ Entry number: _____

Q4 [11 marks] CRDTs

Let us say there is a huge lecture hall room in which we are going to conduct all the major examinations. We want to assign TAs to invigilation duties. The only operations we want to support are `request(ta_entry_num, start_time, stop_time)` and `get(ta_entry_num) -> list[start_time, stop_time]`. These operations may go to different servers which might be temporarily partitioned from one another.

Q4.1 [3 marks] Suggest a *state-based* CRDT to implement these operations.

Hint: `start_time` and `stop_time` are in the unit of hours like 1pm to 3pm, so we need not worry about clock drift.

Q4.2 [3 marks] Justify why your system provides strong eventual consistency.

Q4.3 [2 marks] Give an example of an observable history with your system that is NOT linearizable but exhibits strong eventual consistency.

Name: _____ Entry number: _____

Q4.4 [3 marks] Now let us say we want to further support `busy(ta_entry_num, start_time, stop_time)` where the TA can mark themselves as busy. For example, let us say we call `busy(1234, 1pm, 2pm)` and `request(1234, 12:30pm, 3pm)`. After convergence, `get(1234)` returns `[(12:30pm, 1pm), (2pm, 3pm)]`.

Q5 [4 marks] Distributed transactions

Q5.1 [1 mark] [True/False] When read-write transactions are not conflicting, optimistic concurrency control is expected to be faster than pessimistic concurrency control.

_____

Q5.2 [3 mark] In a two-phase commit, why do transaction participants need to write-ahead prepare yes into their disk before responding with a yes to the prepare request from transaction coordinator. Use a concrete example to explain what goes wrong if a participant does not write prepare yes to disk.

Name: _____ Entry number: _____

This page is intentionally left blank. You may use it for rough work.

Name: _____ Entry number: _____

Q6 [5 marks] Chain replication

Let us say that we have 5 servers maintaining a linearizable key-value store with 1 million keys. Our key-value store only supports `get(key)->val` and `put(key, val)` operations. Further assume that each pair of servers has identical network bandwidth and latency.

CRAQ reduces the load on the tail server by letting non-tail servers serve reads. This also improves locality of reads by letting clients read from a closer server. But CRAQ needs to maintain multiple versions of each key.

[5 marks] Let us say we don't care about improving locality and we do not want to maintain multiple versions of each key due to its implementation complexity. Suggest a scheme to distribute read load across the servers while maintaining linearizability and 5x replication of the key-value store.

Q7 [4 marks] Zookeeper

In Zookeeper, every read and write returns a "zxid" which is sent by the client in the next request. The server is allowed to respond only after its commitIndex is greater than the

request's zxid. Let us say we change this behaviour such that only the writes return a zxid; reads do not return a zxid. In each request, the client sends the largest zxid it knows about.

[4 marks] Show a history that can be observed by clients in this modified Zookeeper, but that canNOT be observed in the original Zookeeper. Justify why this history is now observable.

Q8 [7 marks] Raft

Let us say we modify Raft as follows: each server votes for the *second* vote request. Hence, a candidate may not vote for itself as soon as it becomes a candidate. If the candidate had already received a requestVote RPC, it will vote for itself. Followers cast their vote to the second requestVote RPC that they receive.

Q8.1 [2 marks] Does this change still uphold Election Safety, i.e, at most one leader can be elected in a given term? Justify your answer.

Q8.2 [3 marks] Remember that in the original Raft, we could have a pathological sequence of events where a leader never gets elected even though a majority of servers are able to talk to each other. Original Raft solved this by randomising election timeouts. Do we still need to randomise election timeouts with our change to voting mechanism? Justify your answer.

Name: _____ Entry number: _____

Q8.3 [2 marks] Is this mechanism expected to take longer than the original Raft election mechanism to elect a new leader? Justify your answer.

This area is intentionally left blank. You may use it for rough work.