Design and Implementation of Distributed Applications

Sample Questions
v.3.0

July 1, 2024

Abstract

Some of these questions have been extract from previous exams of DIDA (for the subjects still covered this year). Other questions are new and aim to illustrate the question that may appear in the exam.

Chapter 1

Questions

Paxos.

Consider a Paxos-like algorithm with 3 proposers, 3 acceptors, and 3 learners (in fact, we may have just 3 nodes that act as proposers, acceptors and learners, but this is not relevant for answering the question below).

Consider an execution where the acceptors have the following state:

- Acceptor a_1 has promised to proposer p_1 with timestamp 1 and later has accepted value X from proposer p_1 with timestamp 1.
- Acceptor a_2 has promised to proposer p_2 with timestamp 2 and later has accepted value Y from proposer p_2 with timestamp 2.
- Acceptor a_3 has promised to proposer p_1 with timestamp 1 and later has promised to proposer p_2 with timestamp 2 but did not receive an ACCEPT! from either p_1 or p_2.

§

Question 1 Assume that, at this point, acceptor a_2 receives ACCEPT! (1, X) from proposer p_1. What is the answer of a_2 to p_1?

§

Assume that, at this point, proposer p_3 attempts to propose value Z with timestamp 3. Assume that p_3 runs alone (i.e., acceptors a_1, a_2 and a_3 do not receive messages from proposer p_1 or p_2 while p_3 is running).

Question 2 Can, in this execution, proposer p_3 send ACCEPT! (3, X)? Justify you answer.

Question 3 Can, in this execution, proposer p_3 send ACCEPT! (3, Y)? Justify you answer.

Question 4 Can, in this execution, proposer p_3 send ACCEPT! (3, Z)? Justify you answer.

§

§

Consider a Paxos-like algorithm with 3 proposers, 3 acceptors, and 3 learners (in fact, we may have just 3 nodes act as proposers, acceptors and learners, but this is not relevant for answering the question below). Initially, each proposer is prepared to propose a different value to consensus, namely p_1 wants to propose A, p_2 wants to propose B, and p_3 wants to propose C.

Consider the case where proposer p_1 starts acting as a leader, using timestamp 1, and sends a PREPARE(1) message to all acceptors in an attempt to commit value A. Since it is the first proposer it gets back a promise from acceptors a_1, a_2 and a_3 (with a null value, because there was no previous proposal). Then p_1 sends a ACCEPT! (1, A) to all acceptors. Assume that ACCEPT! (1, A) is received and accepted by acceptor a_1, but not by acceptor a_2 and a_3. Assume that, at this point, the link between proposer p_1 and acceptors a_2 and a_3 becomes slow, and the ACCEPT! message is delayed indefinitely.

Now proposer p_2 suspects that p_1 may have failed and decides to become a leader using timestamp 2, and sends a PREPARE(2) message to all acceptors in an attempt to commit value B.

Question 5 Can, in this execution, proposer p_2 send ACCEPT! (2, A)? Justify you answer.

Question 6 Can, in this execution, proposer p_2 send ACCEPT! (2, B)? Justify you answer.

§

Consider now a different scenario where proposer p_1 starts acting as a leader, using timestamp 1 but very slow and is not able to send the PREPARE message before proposer p_2 suspects that p_1 has failed.

In this case, p_2 starts running alone. send the PREPARE(2) messages, gets a quorum of PROMISE(2,$\perp$) messages, and sends a ACCEPT! (2, B) message to commit value B. Assume that acceptors a_2 and a_3 receive the message and send ACCEPTED (2, B) to the learners.

Assume that, at this point p_1 "wakes up" and sends the PREPARE(1) to all acceptors. Assume that concurrently, p_3 suspects that p_2 may have failed and decides to become a leader using timestamp 3, and sends a PREPARE(3) message to all acceptors in an attempt to commit value C.

Question 7 In this case, what is going to happen to proposer p_1?

Question 8 In this case, what is going to happen to proposer p_3?

§

Consider now a system where nodes need to execute many instances of consensus in sequence, for instance, to implement state-machine replication. During instances $1, \ldots, i$ process p_1 plays the role of the leader but, before executing instance $i+1$, process p_1 fails. Process p_2 suspects the failure of p_1 and decides to become the leader for instances $i+1, \ldots, n$. The Multi-Paxos variant of Paxos allows p_2 to execute an optimisation that will allow consensus executions for instances $i+2, \ldots, n$ to execute faster.

Question 9 Briefly explain in what consists that optimisation.

§

Question 10 Consider a fault-free run of the Paxos algorithm, without the Multi-paxos optimisation. What is the minimum latency (in terms of number of communication steps) since the moment in which a request is received by the current leader and the moment in which the request is decided by the last process? What is the corresponding asymptotic message complexity?

Question 11 Consider a fault-free run of the Paxos algorithm, using the Multi-paxos optimisation. Which phase of Paxos does this optimisation avoids? How many communication steps are saved by this optimisation?

§

Consider a set of nodes $\{p_1, p_2, p_3\}$ running multi-paxos. Each node is a proposer, an acceptor and a learner. Assume that node p_1 was running as a leader until consensus number 11 (using ballot number 1) and is now suspected by p_2 that will attempt to make progress for consensus $[12, \infty]$ using ballot number 2. At this point, process p_2 has a set of commands that still need to be ordered namely $\{F, G, H\}$. Process p_2 is able to get promises from p_2 and p_3 and collects the following information from the promises of acceptors $\{p_1, p_2, p_3\}$, where a tuple includes de following information: $\langle \text{consensus\_instance}, \text{command}, \text{ballot\_number} \rangle$

p_2	p_3
$\langle 12, \perp, 0 \rangle$	$\langle 12, F, 1 \rangle$
$\langle 13, \perp, 0 \rangle$	$\langle 13, \perp, 1 \rangle$
$\langle 14, I, 1 \rangle$	$\langle 14, \perp, 1 \rangle$
$\langle [15, \infty], \perp, 0 \rangle$	$\langle [15, \infty], \perp, 0 \rangle$

Question 12 What is the content of the next ACCEPT! messages sent by p_2?

Coordination Services.

Consider the coordination service implemented by Google and known as Chubby. Consider a replicated service X that relies on primary-backup to keep replicas consistent. To avoid the problem of having two primaries running at the same time, X uses chubby to elect the primary. For this purpose, X uses the lock service offered by Chubby. The primary of X is the process that manages to grab and hold a lock in a given Chubby file. Consider that service X is maintained by replicas x_1, x_2, x_3, x_4 and x_5. Assume that x_1 is the current primary for X and that it owns the lock in the Chubby service.

Question 13 Assume that x_1 becomes disconnected from the network before releasing the lock. What mechanism allows another node x_2 to become the primary?

Question 14 Assume that x_1 is not disconnected from the network but, instead, voluntarily releases the lock. The Chubby service will invalidate the cache in the other clients $x_2 \ldots x_5$. Assume that x_5 is disconnected, preventing its cache to be invalidated, How does Chubby ensures strict consistency in this case?

Question 15 What prevents X from stop working if a Chubby server crashes?

§

Zookeeper is a replicated coordination services that allows read operation to be executed in any replica. As a result, reads can be served by a stale replica and read "from the past".

Question 16 The authors state that a client that needs linearisable reads can always perform a "null" update, before reading. Why is this a solution?

§

Consider the coordination services addressed in classes, namely Chubby and ZooKeeper.

Question 17 Which services offer linearizable updates? And linearizable reads?

§

In both Chubby and ZooKeper, the coordination services are offered by a set of replicas that run a Paxos-like algorithm. Assume that a client needs to read a file from the servers (the client does not have the value cached).

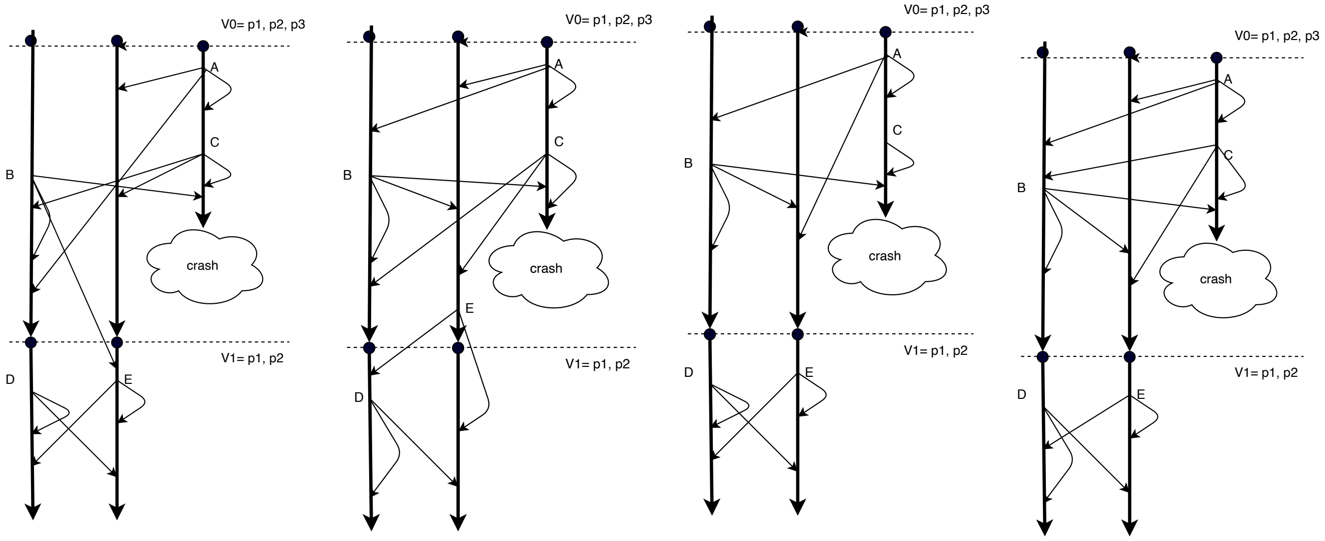
Question 18 In Chubby, can the client read from any replica or just from the primary?

Question 19 In ZooKeeper, can the client read from any replica or just from the primary?

Question 20 Assume a set of clients that want to use Chubby to elect a coordinator. What kind of support is offered by Chubby to support this task?

View Synchrony.

Consider the different executions depicted in Figure 1.1, that illustrate the communication in a group of 3 processes (p_1, p_2, and p_3) using a broadcast primitive. Letters represent message delivery events.



(a) Run 1.

(b) Run 2.

(c) Run 3.

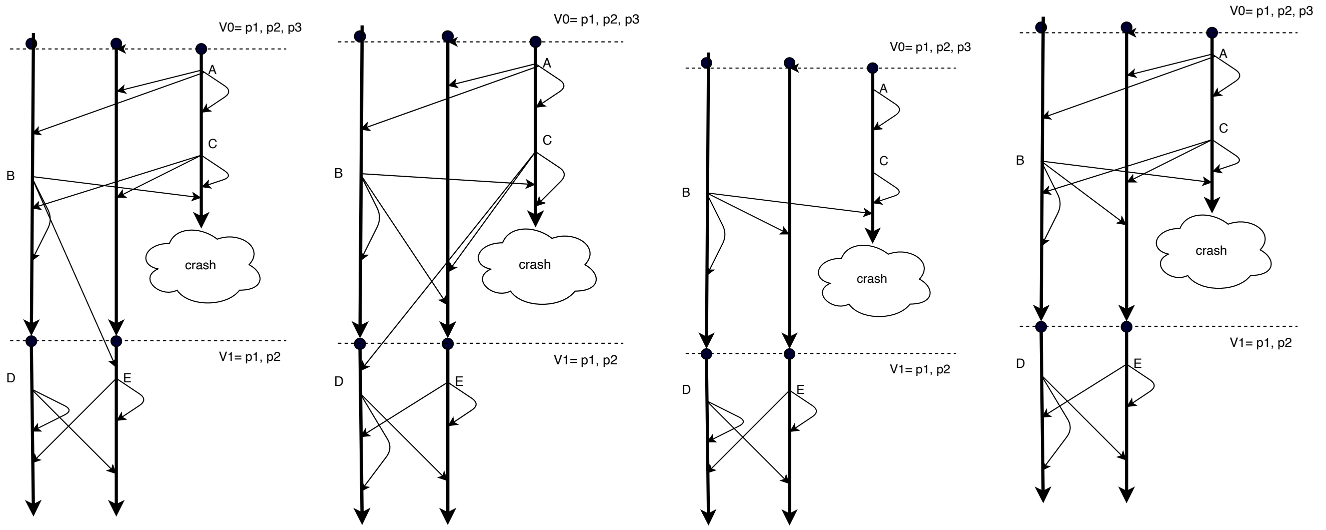
(d) Run 4.

Figure 1.1: Group communication

Question 21 Which runs violate view synchrony? Briefly explain why.

§

Consider the different executions depicted in Figure 1.2, that illustrate the communication in a group of 3 processes (p_1, p_2, and p_3) using a broadcast primitive. Letters represent message delivery events.



(a) Run 1.

(b) Run 2.

(c) Run 3.

(d) Run 4.

Figure 1.2: Group communication

Question 22 Which runs violate uniform reliable broadcast? Briefly explain why.

Question 23 Which runs violate view synchrony? Briefly explain why.

Consider a system that offers view-synchrony and uniform reliable broadcast that respects FIFO order (note that the original ISIS system, did not offer uniform reliable broadcast, only regular reliable broadcast). On top of this system you implement a primary backup system, where clients send request to all replicas and receive replies from all replicas, that works as follows:

```
when new_view (view)
    cleanup ();
    primary := lowest_id (view)

when receive request R from client
    requests_pending := requests_pending ⊕ R

when I am the primary and requests_pending≠ ∅
    R = remove_first (requests_pending)
    ⟨state-update, response⟩ = execute (R);
    vs-uniform-send (R, ⟨state-update, response⟩ ) to all members of the view

when vs-uniform-delivery (R, ⟨state-update, response⟩ )
    remove_from_list (R, requests_pending)
    apply_update (state-update);
    send_reply_to_client (response);
```

Consider that when a node fails a new view is automatically installed by the view-synchrony layer.

Question 24 Specify the code of the CLEANUP procedure.

Stoppable Paxos.

Consider a classical implementation of state machine replication that uses a sequence of *independent* Paxos instances (instead of a multi-instance protocol running stoppable Paxos). You can reconfigure the state machine by issuing a special RECONFIG command. When this command is accepted at some Paxos instance (say, consensus instance n), no more commands should be accepted in future instances (i.e., instance $n+1$ and higher) using the same configuration. Instead, future commands should be processed by the new Paxos configuration.

Assume a given configuration of the state machine replication that consists of processes $C_1 = \{p_1, p_2, p_3\}$, where each process is a proposer, an acceptor, and a learner. Assume that proposer p_2 whats to propose RECONFIG command to reconfigure the state machine to use $C_2 = \{p_2, p_3, p_4\}$.

Question 25 Can p_3 propose some command X to consensus instance n in C_1 without first learning the result of instances $n-1$ and lower?

§

Consider now that the system is using the "delayed stop sign" method with $\alpha = 5$

Question 26 Can p_3 propose some command X to consensus instance n in C_1 without first learning the result of instances $n-1$ and lower?

§

Consider a set of nodes $\{p_1, p_2, p_3\}$ running stoppable Paxos. Each node is a proposer, an acceptor and a learner. Assume that node p_1 was running as a leader until consensus number 11 (using ballot number 1) and is now suspected by p_2 that will attempt to make progress for consensus $[12, \infty]$ using ballot number 2. At this point, process p_2 has a set of commands that still need to be ordered namely $\{F, G, H\}$ (i.e., p_2 is not planing to stop the Paxos configuration). Process p_2 is able to get promises from p_2 and p_3 and collects the following information from the promises of acceptors $\{p_1, p_2, p_3\}$, where a tuple includes de following information: $\langle \text{consensus\_instance}, \text{command}, \text{ballot\_number} \rangle$

p_2	p_3
$\langle 12, \perp, 0 \rangle$	$\langle 12, F, 1 \rangle$
$\langle 13, \perp, 0 \rangle$	$\langle 13, G, 1 \rangle$
$\langle 14, \perp, 0 \rangle$	$\langle 14, \textsc{stop}, 1 \rangle$
$\langle [15, \infty], \perp, 0 \rangle$	$\langle [15, \infty], \perp, 0 \rangle$

Question 27 What is the content of the next ACCEPT! messages sent by p_2?

§

Consider a set of nodes $\{p_1, p_2, p_3\}$ running stoppable Paxos. Each node is a proposer, an acceptor and a learner. Assume that node p_1 was running as a leader until consensus number 11 (using ballot number 1) and is now suspected by p_2 that will attempt to make progress for consensus $[12, \infty]$ using ballot number 5. At this point, process p_2 as an set of commands that still need to be ordered namely $\{\textsc{stop}\}$ (i.e., p_2 wants to stop the Paxos configuration). Process p_2 is able to get promises from p_2 and p_3 and collects the following information from the promises of acceptors $\{p_1, p_2, p_3\}$, where a tuple includes de following information: $\langle \text{consensus\_instance}, \text{command}, \text{ballot\_number} \rangle$

p_2	p_3
$\langle 12, \perp, 0 \rangle$	$\langle 12, F, 1 \rangle$
$\langle 13, \perp, 0 \rangle$	$\langle 13, \textsc{stop}, 1 \rangle$
$\langle 14, \perp, 0 \rangle$	$\langle 14, G, 3 \rangle$
$\langle [15, \infty], \perp, 0 \rangle$	$\langle [15, \infty], \perp, 0 \rangle$

Question 28 What is the content of the next ACCEPT! messages sent by p_2?

§

Consider the following algorithm to reconfigure a replicated state machine (RSM) running a form of multi-paxos in configuration C: Any proposer could send a special command RECONFIG(C'), with a new configuration C'. This command would be accepted in some paxos instance, say instance n. After this, the choice of the command to be accepted in instance $n+1$ would need to be performed by the nodes belonging to configuration C' (and no longer by configuration C).

Question 29 What is the main disadvantage of this method?

Reconfigurable Registers.

Consider a system implementing atomic registers using the ABD algorithm. In this system, writes return after writing in a majority. Needs need to read from a majority, select the most recent value, and write-back the value before returning. Assume that you need to support the reconfiguration of the register replicas. For this purpose, you use a coordination service (for instance ZooKeeper) that ensures that there is a total order on all configurations. ZooKeeper also informa all clients of a configuration change (note however that in ZooKeeper, clients are informed asynchronously, and can get the notifications with some delay).

Assume a system where $C_1 = \{p_1, p_2, p_3\}$ and $C_2 = \{p_2, p_3, p_4\}$.

Assume that each register keeps a tuple with the following format:

$$\langle \text{value}, \text{timestamp}, \text{pointer to the next configuration} \rangle$$

where the "pointer to the next configuration" is "null" if no other configuration exists or $C_{\text{next}} = \{p_i, p_j, \ldots\}$ if a future configuration exists.

The write operation can change the value of the register or the "pointer to the next configuration" .

Question 30 Consider a client c_1 running in configuration C_1 that reads $\langle A, 10, \text{null} \rangle$ from p_1 and $\langle A, 11, C_2 = \{p_2, p_3, p_4\} \rangle$ from p_2. What should client c_1 do?

Question 31 Consider a client c_2 that learns from ZooKeeper configuration C_2 and needs to write some value X on the register. This is the first write operation c_2 will do on the new configuration. What are the steps c_2 needs to execute?

Raft.

§

Consider the Raft algorithm an the sequence of term leaders illustrated in Figure 1.3.

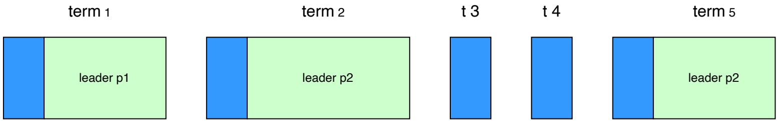


Figure 1.3: Raft Term Leaders

Question 32 Describe a scenario that can cause the sequence illustrated in Figure 1.3.

§

Consider a set of 5 processes running Raft. Figure 1.4 illustrates the content of the log of each process at a given point in time, where each entry is a tuple “(command, term)”.

P1	A(1)	B(1)	C(1)	H(4)	D(4)		
P2	A(1)	B(1)	C(1)	H(4)	E(5)		
P3	A(1)	B(1)	C(1)	F(2)	G(2)		
P4	A(1)	B(1)	C(1)	H(4)	E(5)	F(5)	G(5)
P5	A(1)	B(1)	C(1)	D(1)	E(1)		

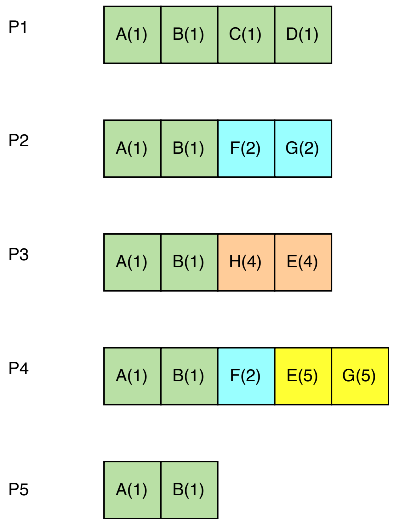
Figure 1.4: Logs of $p_1 \ldots p_5$

Question 33 Enumerate a sequence of term leaders that is consistent with the state depicted in Figure 1.4 .

Question 34 Assume that the leader of term 5 fails. Which processes may become leaders at this point?

§

Consider a set of 5 processes running Raft. Figure 1.5 illustrates the content of the log of each process at a given point in time, where each entry is a tuple “(command, term)”.



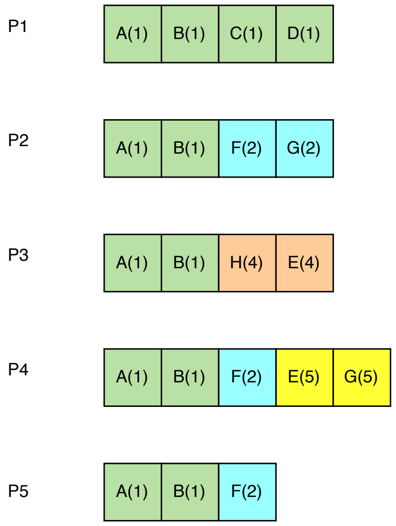
P1	A(1)	B(1)	C(1)	D(1)	
P2	A(1)	B(1)	F(2)	G(2)	
P3	A(1)	B(1)	H(4)	E(4)	
P4	A(1)	B(1)	F(2)	E(5)	G(5)
P5	A(1)	B(1)			

Figure 1.5: Logs of $p_1 \ldots p_5$

Question 35 Describe a scenario that allows p_4 to become the leader of term 5.

§

Assume that, after winning the election for term 5, p_4 starts to propagate its log to p_5. As a result, command $F(2)$ is now replicated in a majority of replicas, as illustrated in Figure 1.6. However command $F(2)$ is not yet commited.



P1	A(1)	B(1)	C(1)	D(1)	
P2	A(1)	B(1)	F(2)	G(2)	
P3	A(1)	B(1)	H(4)	E(4)	
P4	A(1)	B(1)	F(2)	E(5)	G(5)
P5	A(1)	B(1)	F(2)		

Figure 1.6: Logs of $p_1 \ldots p_5$

Question 36 Whan can happen to “remove” $F(2)$ from the third position in the log?

Database Replication.

Consider a replicated database with 3 replicas. Consider a simplified model of the reality where every action takes one or multiple time "units" to execute (units are abstract, and can represent several μs in real life). Consider, the following "cost" (in term of time units) of different operations:

Action	Duration (in time units)
Start a transaction at a given replica	1
Execute a transaction at a given replica TA(x)	x
Send a totally order broadcast (TOB)	1
Certify the read/write set of a transaction	1
Commit or Abort a transaction	1

Question 37 Consider a system using state machine replication, executing two transactions TA(3) e TB(1). TA is submitted at time 0 on replica R_1 and TB is submitted at time 1 on replica R_2. Assume that TA and TB conflict and that one needs to abort if they execute concurrently. Complete the following table, that describes what happens in the system from the perspective of an external observer.

Database Replication:

		1	2	3	4	5
Question 37	R_1	Start TA				
	R_2	-	Start TB			
	R_3	-	-			
	network	-				

Database Replication:

		6	7	8	9	10
Question 37	R_1					
	R_2					
	R_3					
	network					

Question 38 Consider a system using multi-master replication, with optimistic concurrency control and global certification, executing two transactions TA(3) e TB(1). TA is submitted at time 0 on replica R_1 and TB is submitted ar time 1 on replica R_2. Assume that TA and TB conflit and that one needs to abort if they execute concurrently. Complete the following table, that describes what happens in the system from the perspective of an external observer.

Database Replication:

		1	2	3	4	5
Question 38	R_1	Start TA				
	R_2	-	Start TB			
	R_3					
	network					

Database Replication:

		6	7	8	9	10
Question 38	R_1					
	R_2					
	R_3					
	network					

§

Consider a replicated database with 3 replicas. Consider a simplified model of the reality where every action takes one or multiple time "units" to execute (units are abstract, and can represent several μs in real life). Consider, the following "cost" (in term of time units) of different operations:

Action	Duration (in time units)
Start a transaction at a given replica	1
Execute a transaction at a given replica TA(x)	x
Send a totally order broadcast (TOB)	1
Send an Uniform Reliable Broadcast (URB)	1
Certify the read/write set of a transaction	1
Commit or Abort a transaction	1

Question 39 Consider a system using multi-master replication, with optimistic concurrency control and local certification (a variant of multi-master where only the write set needs to be sent in the network and where only the source of the transaction can certify the outcome), executing two transactions TA(3) e TB(1). TA is submitted at time 0 on replica R_1 and TB is submitted ar time 1 on replica R_2. Assume that TA and TB conflit and that one needs to abort if they execute concurrently. Complete the following table, that describes what happens in the system from the perspective of an external observer.

Database Replication:

		1	2	3	4	5
Question 39	R_1 R_2 R_3 network	Start TA -	 Start TB			

Database Replication:

		6	7	8	9	10
Question 39	R_1 R_2 R_3 network					

Spanner.

Spanner is a transactional distributed database. The database is partitioned in multiple tablets, where different tablets may be stored and managed by different groups of replicas. A transaction can change items in multiple tablets. In this case, a two-phase commit protocol is executed to serialize the transaction across tablets.

§

One know problem of the classical two-phase commit protocol is that it is blocking: if the coordinator fails after sending a prepare message, the participants may block until the coordinator recovers.

Question 40 How does Spanner circumvent the blocking problem described above?

§

Spanner offers external consistency (also know has linearizability).

Question 41 What is the difference between linearizability and serializability?

Question 42 What service is used by Spanner to enforce linearizability in an efficient way?

§

Consider the leader of a Spanner shard. Consider that the leader keeps 3 versions of a given key K, namely:

- $\langle K, A, 100, \text{committed}\rangle$,
- $\langle K, B, 200, \text{committed}\rangle$,
- $\langle K, C, 300, \text{prepared}\rangle$.

Each version is a tuple: $\langle \text{key}, \text{value}, \text{timestamp}, \text{status: committed/ prepared}\rangle$.

In a version that is committed, the timestamp is the final timestamp assigned to the transaction by the coordinator. In a prepared version, the timestamp is the vote of the local shard. Note that timestamps are simplified (Spanner uses synchronised clocks and not logical clocks).

Question 43 Assume that a transaction reading from snapshot 150 attempts to read key K. What value is returned?

Question 44 Assume that a transaction reading from snapshot 250 attempts to read key K. What value is returned?

Question 45 Assume that a transaction reading from snapshot 350 attempts to read key K. What value is returned?

TCC.

Consider the Cure system that offers Transaction Causal Consistency. Consider a deployment using 3 datacenters and a database divided in 3 shards. Consider the state of each datacenter as depicted in Figure 1.7 (note that cures uses hybrid clocks and values correspond to physical clocks; here we use logical clocks to simplify the notation). In the following, when denote a transaction committed with timestamp 10, simply as "T10".

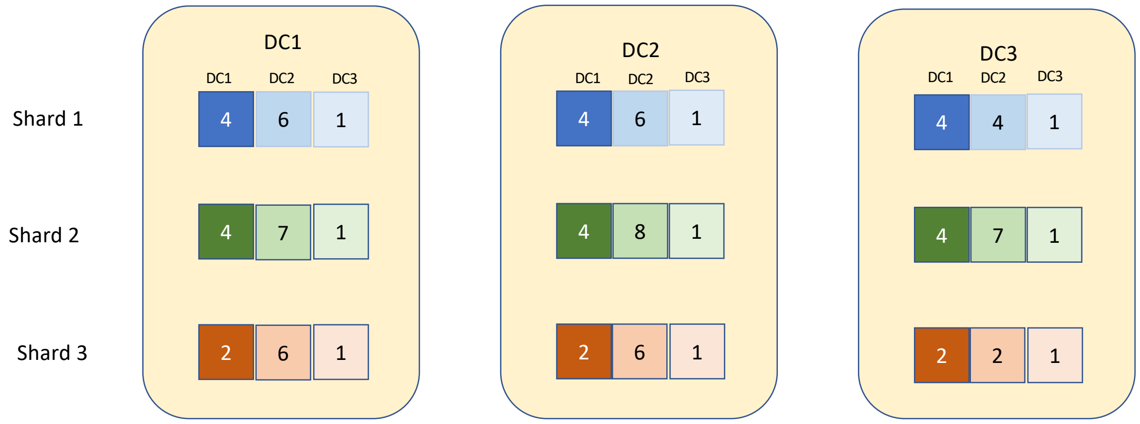


Figure 1.7: Cure

Question 46 Assume that a client executes a read transaction on DC_3 that reads all shards. Will that client observe the effect of transaction T7 from DC_2?

Question 47 Assume that a client starts a transaction on DC_2. What will be vector clock that defines the read snapshot of that transaction?

Question 48 Assume that a client commits a transaction T10 on DC_1 that updates shard 1 (only). Assume that the vector clock associated with that transaction is $[10, 6, 1]$. Shard 1 send the update associated with T10 do DC_3. Can this update be applied immediately on DC_3?

Question 49 TCC allows multiple concurrent transactions to update the same objects and still commit (unlike strong consistency models, that allow only one of the conflicting transaction to commit). What data structures are used in Cure to prevent conflicting updates to simply overwrite the others?

§

Assume a key-value store with just 4 keys, K_1, K_2, K_3, and K_4. Assume the following sequence of transactions:

- Transaction T_A writes all keys.
- Transaction T_B reads the values written by T_A and writes on keys K_1 and K_4.
- Transaction T_C reads the values written by T_A and writes on keys K_1 and K_2.
- Transaction T_D reads the values written by T_C and writes on keys K_1 and K_3.

Assume that we can distinguish each version of the key, using the identifier of the transaction that committed the corresponding value. For instance $\langle K_1, T_A \rangle$ is the value of key k_1 written by transaction T_A. After the sequence above, each key as the following versions:

K_1	$\langle K_1, T_A \rangle$,	$\langle K_1, T_B \rangle$	$\langle K_1, T_C \rangle$	$\langle K_1, T_D \rangle$
K_2	$\langle K_2, T_A \rangle$,		$\langle K_2, T_C \rangle$	
K_3	$\langle K_3, T_A \rangle$,			$\langle K_3, T_D \rangle$
K_4	$\langle K_4, T_A \rangle$,	$\langle K_4, T_B \rangle$		

Question 50 (*1.0 point*) Consider now a transaction T_E that reads all keys. For each set of values read by transaction T_E presented below, state which are legal under TCC.

Scenario 1, T_E reads:	$\langle K_1, T_A \rangle$, $\langle K_2, T_A \rangle$, $\langle K_3, T_A \rangle$, $\langle K_4, T_A \rangle$
Scenario 2, T_E reads:	$\langle K_1, T_A \rangle$, $\langle K_2, T_A \rangle$, $\langle K_3, T_D \rangle$, $\langle K_4, T_B \rangle$
Scenario 3, T_E reads:	$\langle K_1, T_B \rangle$, $\langle K_2, T_A \rangle$, $\langle K_3, T_A \rangle$, $\langle K_4, T_B \rangle$
Scenario 4, T_E reads:	$\langle K_1, T_B \rangle$, $\langle K_2, T_A \rangle$, $\langle K_3, T_D \rangle$, $\langle K_4, T_B \rangle$
Scenario 5, T_E reads:	$\langle K_1, T_B \rangle$, $\langle K_2, T_C \rangle$, $\langle K_3, T_D \rangle$, $\langle K_4, T_B \rangle$
Scenario 6, T_E reads:	$\langle K_1, T_C \rangle$, $\langle K_2, T_C \rangle$, $\langle K_3, T_A \rangle$, $\langle K_4, T_B \rangle$
Scenario 7, T_E reads:	$\langle K_1, T_C \rangle$, $\langle K_2, T_C \rangle$, $\langle K_3, T_D \rangle$, $\langle K_4, T_B \rangle$
Scenario 8, T_E reads:	$\langle K_1, T_C \rangle$, $\langle K_2, T_C \rangle$, $\langle K_3, T_A \rangle$, $\langle K_4, T_A \rangle$
Scenario 9, T_E reads:	$\langle K_1, T_D \rangle$, $\langle K_2, T_A \rangle$, $\langle K_3, T_D \rangle$, $\langle K_4, T_A \rangle$
Scenario 10, T_E reads:	$\langle K_1, T_D \rangle$, $\langle K_2, T_C \rangle$, $\langle K_3, T_D \rangle$, $\langle K_4, T_B \rangle$

Assume now that each transaction is assigned an unique timestamp at commit time, and that all values written by that transaction are tagged with the commit time of the transaction. Assume a datacenter D that receives updates from remote datacenters asynchronously, using gossip. Updates for different keys are propagated independently of each other (i.e., updates from the same transaction on different keys can be received at different times). Consider that, at a given point, datacenter D has received the following set of updates from remote datacenters:

K_1	$\langle K_1, 5 \rangle$,		$\langle K_1, 15 \rangle$	$\langle K_1, 20 \rangle$	
K_2	$\langle K_2, 5 \rangle$,		$\langle K_2, 15 \rangle$		
K_3	$\langle K_3, 5 \rangle$,		$\langle K_3, 15 \rangle$	$\langle K_3, 20 \rangle$	
K_4	$\langle K_4, 5 \rangle$,	$\langle K_4, 10 \rangle$			$\langle K_4, 30 \rangle$

Question 51 (*1.0 point*) Consider now a transaction T_E that reads keys K_1 and K_4 at datacenter D. Which are the most recent values that can be returned without risking violating TCC?

P2P.

§

Consider the peer-to-peer system Chord, using identifiers/keys with 4 bits. There are five active nodes that have the following identifiers: 2, 3, 7, 12, 14.

Question 52 Assign to nodes the documents with the following keys: 1, 8, 9, 13.

Question 53 Provide the finger table of the node with ID = 2.

§

Consider the peer-to-peer system Chord, using identifiers/keys with 4 bits. There are five active nodes that have the following identifiers: 2, 3, 6, 11, 12.

Question 54 Assign to nodes the documents with the following keys: 1, 4, 11, 15.

Question 55 Provide the finger table of the node with ID = 4.

§

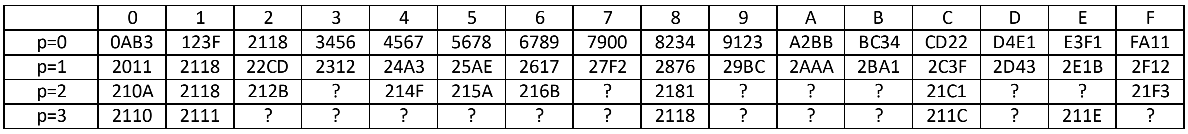
Consider the peer-to-peer system Chord, using identifiers/keys with 8 bits. There are six active nodes that have the following identifiers: 24, 64, 102, 144, 174, 220.

Question 56 Provide the finger table of the node with ID = 102.
Assume node 64 owns the objects with following keys: 27, 29, 35, 55. Now assume that a node with ide 30 also joins the network.

Question 57 What is the distribution of the keys previously owned by node 64 after node 30 joins?

§

Consider the peer-to-peer system Pastry. Assume that, for simplicity, the address space is composed by 4 hexadecimal digits, spanning the range [0000-FFFF]. Assume that one of the nodes of the system, say n, contains the following routing table (Fig. 1.8), where p denotes the size of the common prefix.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
p=0	0AB3	123F	2118	3456	4567	5678	6789	7900	8234	9123	A2BB	BC34	CD22	D4E1	E3F1	FA11
p=1	2011	2118	22CD	2312	24A3	25AE	2617	27F2	2876	29BC	2AAA	2BA1	2C3F	2D43	2E1B	2F12
p=2	210A	2118	212B	?	214F	215A	216B	?	2181	?	?	?	21C1	?	?	21F3
p=3	2110	2111	?	?	?	?	?	?	2118	?	?	?	211C	?	211E	?

Figure 1.8: Pastry routing table.

Question 58 Which of the the following 4 processes can have the the routing table above: `0AB3`, `123F`, `2118`, or `3456`? Justify your answer.

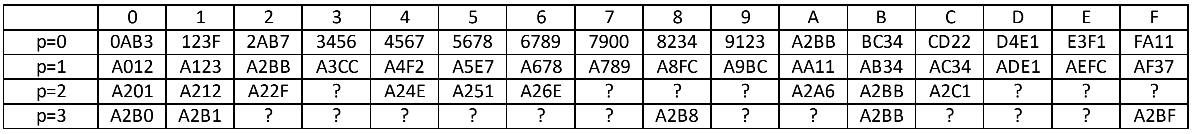
Question 59 If n needs to route a message to node `24A5`, which node is n going to contact?

Question 60 If n needs to route a message to node `214C`, which node is n going to contact?

Question 61 During the process of node `214C` joining the Pastry network, it routes the joining message through node n. Node n will help node `214C` to build its new routing table. Which line of the routing table of node n is sent to node `214C`?

§

Consider the peer-to-peer system Pastry. Assume that, for simplicity, the address space is composed by 4 hexadecimal digits, spanning the range [0000-FFFF]. Assume that one of the nodes of the system, say n, contains the following routing table (Fig. 1.9), where p denotes the size of the common prefix.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
p=0	0AB3	123F	2AB7	3456	4567	5678	6789	7900	8234	9123	A2BB	BC34	CD22	D4E1	E3F1	FA11
p=1	A012	A123	A2BB	A3CC	A4F2	A5E7	A678	A789	A8FC	A9BC	AA11	AB34	AC34	ADE1	AEFC	AF37
p=2	A201	A212	A22F	?	A24E	A251	A26E	?	?	?	A2A6	A2BB	A2C1	?	?	?
p=3	A2B0	A2B1	?	?	?	?	?	?	A2B8	?	?	A2BB	?	?	?	A2BF

Figure 1.9: Pastry routing table.

Question 62 Which of the the following 3 processes can have the the routing table above: 0AB3, A2BB, or FA11? Justify your answer.

Question 63 If n needs to route a message to node 355C, which node is n going to contact?

Question 64 If n needs to route a message to node A3F0, which node is n going to contact?

Question 65 During the process of node $A2EF$ joining the Pastry network, it routes the joining message through node n. Node n will help node $A2EF$ to build its new routing table. Which line of the routing table of node n is sent to node $A2EF$?

Dynamo.

Question 66 Dynamo uses "Virtual nodes". Briefly explain what are virtual nodes and what is the main purpose of using virtual nodes.

Question 67 A Dynamo deployment can have hundreds of nodes. In Dynamo, when an administrator adds or removes a node, does the system needs to contact immediately all these nodes to update their views? If not, how are views updated?

Consider the partial order of updates to an object in Dynamo depicted in Fig. 1.10.

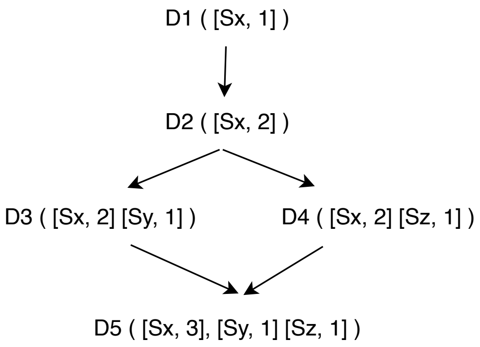


Figure 1.10: Partial order of versions.

Question 68 Describe a sequence of updates that can generate this graph.

Chapter 2

Solutions

.

Paxos:

Question 1	It will not accept the proposal from p_1 because it has promised to p_2
Question 2	Yes, if it gets a promise from a_1 and a_3
Question 3	Yes, if it gets a promise from a_2 and a_3
Question 4	No. Either a_1 or a_2 belong to a majority, and p_3 will adopt the most recent value it observes

§

Paxos::

Question 5	Yes, if p_2 receives PROMISE(2, $\langle 1, A\rangle$) from a_1
Question 6	Yes, if p_2 receives PROMISE(2, $\perp$) from a_2 and a_3
Question 7	Proposer p_1 needs to abort and may try again with timestamp 4 or larger.
Question 8	p_3 will receive at least one PROMISE(3, $\langle 2, B\rangle$) and will commit B
Question 9	The new leader will still performs both phases for all rounds. However, in round $i+1$, it will send immediately PREPARE for rounds $i+2, \ldots, n$ It does so, even before knowing what values it will propose in the corresponding accept messages. If the PREPARE is accepted, later it can send just the ACCEPT! message.

§

Paxos:

Question 10	Minimum Latency: 4 communication steps Message Complexity: $O(n^2)$ 1 RTT to end Phase 1, 1 RTT to finalize Phase 2, assuming acceptors broadcast Accepted message to all learners
Question 11	Phase 1. 1 RTT (2 communication steps) can be saved.

§

Paxos:

Question 12	$\langle$ACCEPT!$, 12, F, 2\rangle$ $\langle$ACCEPT!$, 13, G, 2\rangle$ $\langle$ACCEPT!$, 14, I, 2\rangle$ $\langle$ACCEPT!$, 15, H, 2\rangle$

§

Chubby:

Question 13	x_1 has a lease on the lock. it needs to refresh the lease periodically in order to maintain the lock if x_1 fails to renew the lock before the lease expires, it needs to release the lock voluntarily
Question 14	The cache of client x_5 is valid only for the lease period Chubby will wait for an acknowledgment until the lease expires if x_5 fails to renew the cache lease before it expires, it needs invalidate its own cache
Question 15	The Chubby service is provided by multiples replicas, which are kept consistent using Paxos

§

Zookeeper:

Question 16	Zookeeper does provide linearisable updates It also guarantees “read-your-writes” by doing a “null” update, thanks to linearisable writes, the client is sure that its own “null” update is executed in the future of all previous updates from “read-your-writes” the client will see its own “null” update (and all updates before that) in practice, this makes linearisable reads as expensive as updates

§

Chubby vs ZooKeeper	
Question 17	Chubby provides linearizable updates and linearizable reads. ZooKeeper only offers linearizable updates.
Question 18	Just from the primary (to ensure linearizable reads)
Question 19	From any replica.
Question 20	Chubby supports locks. Also, Chubby can release a lock if the client owning the lock fails to renew its lease. If all clients attempt to get the lock, the lock owner is "elected" as coordinator.

§

View Synchrony:	
Question 21	Run 1, Run 2

§

View Synchrony:	
Question 22	Run 3 (A and C delivered at p_3 but not p_1 and p_2)
Question 23	Run 1 (B delivered before the view at p_1 and after the view at p_2) Run 2 (C delivered before the view at p_2 and after the view at p_1)

§

View Synchrony:	
Question 24	CLEANUP () { }

§

Reconfigurable Paxos:	
Question 25	no, because it may happen that command $n-1$ is RECONFIG
Question 26	yes, at long it has learned the output of instance $n-5$ and lower

§

Stoppable Paxos:	
Question 27	$\langle$ACCEPT!$, 12, F, 2\rangle$ $\langle$ACCEPT!$, 13, G, 2\rangle$ $\langle$ACCEPT!$, 14,$ STOP$, 2\rangle$

§

Stoppable Paxos:	
Question 28	$\langle$ACCEPT!$, 12, F, 5\rangle$ $\langle$ACCEPT!$, 13,$ NO-OP$, 5\rangle$ $\langle$ACCEPT!$, 14, G, 5\rangle$ $\langle$ACCEPT!$, 15,$ STOP$, 5\rangle$

§

Reconfiguration of Paxos	
Question 29	Processes cannot start executing instance $n+1$ without knowing the accepted value of instance n; This limits the paralelism in the system and degrades performance

§

Registers:	
Question 30	Client c_1 will need to read from a majority of C_2
Question 31	Client c_2 will need to write the pointer to C_2 on a majority of nodes from C_1 and then write X on a majority of nodes of C_2

§

Raft:	
Question 32	Process p_1 wins the election for term 1 Process p_1 is suspected, p_2 wins the election for term 2 p_2 is suspected but no process wins the election for term 3 no process wins the election for term 4 p_2 recovers and wins the election for term 5

§

Raft:

Question 33		Process p_5 is the leader for term 1 Process p_3 is the leader for term 2 no process wins the election for term 3 Process p_1 is the leader for term 4 Process p_4 is the leader for term 5
Question 34		Process p_1 with the votes of p_3 and p_5 Process p_2 with the votes of p_1, p_3 and p_5 Process p_4 (all other processes can vote for it)

§

Raft:

Question 35	Process p_4 can win the election for term 5 with the votes of p_1 and p_5
Question 36	If p_4 fails, p_3 can win the election for term 6 and revert "$F(2)$" to "$H(4)$"

Database Replication:

		1	2	3	4	5
Question 37	R_1	Start TA	-	Exec TA	Exec TA	Exec TA
	R_2	-	Start TB	Exec TA	Exec TA	Exec TA
	R_3	-	-	Exec TA	Exec TA	Exec TA
	network	-	TOB (TA)	TOB (TB)	-	-

Database Replication:

		6	7	8	9	10
Question 37	R_1	Commit TA	Exec TB	Commit TB	-	-
	R_2	Commit TA	Exec TB	Commit TB	-	-
	R_3	Commit TA	Exec TB	Commit TB	-	-
	network	-	-	-	-	-

§

Database Replication:

		1	2	3	4	5
	R_1	Start TA	Exec TA	Exec TA	Exec TA	Certify TB
Question 38	R_2	-	Start TB	Exec TB	-	Certify TB
	R_3	-	-	-	-	Certify TB
	network	-	-	-	TOB (TB)	TOB (TA)

Database Replication:

		6	7	8	9	10
	R_1	Commit TB	Certify TA	Abort TA	-	-
Question 38	R_2	Commit TB	Certify TA	Abort TA		
	R_3	Commit TB	Certify TA	Abort TA	-	-
	network	-	-	-	-	-

§

Database Replication:

		1	2	3	4	5
	R_1	Start TA	Exec TA	Exec TA	Exec TA	-
Question 39	R_2	-	Start TB	Exec TB	-	Certify TB
	R_3	-	-	-	-	-
	network	-	-	-	TOB (TB)	TOB (TA)

Database Replication:

		6	7	8	9	10
	R_1	-	Commit TB	Certify TA	Abort TA	
Question 39	R_2	Commit TB	-	-	-	Abort TA
	R_3	-	Commit TB	-	-	Abort TA
	network	URB (Commit TB)	-	-	URB (Abort TA)	-

§

Spanner:

Question 40	In Spanner, each participant is replicated using Paxos If a replica of a given participant fails, the participant can continue to operate. Paxos will elect a new replica as the leader for that participant. The system will only block if a majority of the replicas of the coordinator fail.
Question 41	Serializability: the execution is equivalent to a serial execution. Let transaction T_b start after transaction T_a commits Linearisation: the execution is equivalent to a serial execution where T_b must be serialised after T_a (prevents transactions from being serialised in the past)
Question 42	A clock synchronisation service/API called TrueTime

§

Spanner:

Question 43	A
Question 44	B
Question 45	Read will block, may return B or C depending on the final commit time of C

§

TCC::

Question 46	No. The stable snapshot at DC_3 is $[2, 2, 1]$. T7 will only become visible when updates with timestamp 7 or larger are received from DC_2 for shards 1 and 3.
Question 47	$[2, 8, 1]$
Question 48	No. T10 has T6 from DC_2 in its causal past. and T6 from DC_2 has not been applied at DC_3 yet.
Question 49	Cure advocates the user of Conflict-free Replicated Datatypes. These are data structures that simplify the execution "merge" operations that can "combine" concurrent updates in a single consistent state

§

TCC:

	Scenario	Valid? (Yes/No)
Question 50	Scenario 1	yes
	Scenario 2	no (if reads $\langle K_3, T_D\rangle$ it must read $\langle K_1, T_D\rangle$)
	Scenario 3	yes
	Scenario 4	no (if reads $\langle K_3, T_D\rangle$ it must read $\langle K_1, T_D\rangle$)
	Scenario 5	no (if reads $\langle K_2, T_C\rangle$ it must read $\langle K_1, T_C\rangle$)
	Scenario 6	yes
	Scenario 7	no (if reads $\langle K_3, T_D\rangle$ it must read $\langle K_1, T_D\rangle$)
	Scenario 8	yes
	Scenario 9	no (T_D depends on T_C, must read $\langle K_2, T_C\rangle$)
	Scenario 10	yes
	key	Value returned
Question 51	K_1	$\langle K_1, 20\rangle$
	K_4	$\langle K_4, 10\rangle$

§

§

Chord

Question 52	Item	Stored By
	1	2
	8	12
	9	12
	13	14
Question 53	Finger	Points To
	0	3
	1	7
	2	7
	3	12

§

Chord

Question 54	Location of files: 1 ⇒ 2 4 ⇒ 6 11 ⇒ 11 15 ⇒ 2
Question 55	Finger table for node 4: 0 ⇒ (4+1=5) ⇒ 6 1 ⇒ (4+2=6) ⇒ 6 2 ⇒ (4+4=8) ⇒ 11 3 ⇒ (4+8=12) ⇒ 12

§

Chord

Question 56	Finger 0	Node 144	Finger 1	Node 144	Finger 2	Node 144	Finger 3	Node 144
	Finger 4	Node 144	Finger 5	Node 144	Finger 6	Node 174	Finger 7	Node 24

Question 57	Node 30 gets keys 27 and 29 Node 64 keeps keys 35 and 55

§

Pastry

Question 58	2118
Question 59	24A3
Question 60	214F
Question 61	3rd line ($p = 2$)

§

Pastry

Question 62	A2BB Only this none will have a row for A2B* in its routing table
Question 63	3456 This is the only node starting with 3*** that n knows
Question 64	A3CC This is the only node starting with A3** that n knows
Question 65	Typically the routing request will be sent via **** → A*** → A2** → A2E* Thus, in the general case A2BB will be the 3rd process contacted It this case it will send row p_2 Note that A2** is the longest prefix shared by A2BB and A2EF (in the unlikely case A2BB is the first process to be contacted, it will send p_0, p_1 and p_2)

§

Dynamo

Question 66	Virtual nodes are a strategy to let a single physical machine host multiple logical ring nodes. The purpose of virtual nodes is to promote a better load balancing among physical nodes
Question 67	Dynamo uses a gossip protocol to propagate membership changes in background

Question 68	Updates D1 and D2 are created at node Sx. Nodes Sy and Sz receive update D2. Node Sy makes update D3 and, concurrently, node Sz makes another update D4 Updates D3 and D4 are propagated to Sx that reconciles them Nodes Sx creates version D5 that merges D3 and D4.

§