Results for question 1.

When does a read operation of a transaction T_c arrive too late and have to be aborted in concurrency control based on timestamp ordering?

When T_c has already written its own version of the object.

when a committed transaction with an earlier timestamp has written the object.

when a transaction with a later timestamp has already written the object.

when an aborted transaction with an earlier timestamp has written the object.

Results for question 2.

Which of the following statements about transactions and concurrency control are correct? Select correct statement(s).

In concurrent control based on timestamp ordering, a write operation of a current transaction T_c conflicts with previous read and write operations done by other transactions, T_i, whose timestamps indicate that they should be later than T_c, i.e., $T_i> T_c$

With the strict two-phase locking, a transaction acquires all locks it needs in the first phase (a "growing phase"), and releases the locks in the second phase (a '"shrinking phase") before the transaction commits or aborts.

A serially equivalent interleaving is an interleaving of the operations of transactions in which the combined effect is the same as if the transactions had been performed one at a time in some order.

In concurrent control based on timestamp ordering, a read operation of a current transaction T_c conflicts with previous read and write operations done by other transactions, T_i, whose timestamps indicate that they should be later than T_c, i.e., $T_i> T_c$

To prevent cascading aborts, a transaction must suspend its commit operation if it has performed a dirty read.

In optimistic concurrent control, backward validation checks a write set of the transaction undergoing validation with read sets of other later transactions, which are still active.

In order to recover from an aborting transaction, a transaction must not commit if it has done a dirty read from a transaction that subsequently aborted.

Results for question 3.

Which of the following statements about distributed transactions are correct? Select correct statement(s).

A phantom deadlock could be detected if during the deadlock detection procedure, one of the transactions in a deadlock cycle that holds a lock meanwhile has released the lock.

A distributed deadlock can be detected by having each of servers, from time to time, to check for cycles in the latest copy of its local wait-for graph. When the server finds a cycle, it makes a unilateral decision on how to resolve the deadlock and which transaction to abort.

If, after having promised to commit in the two-phase commit protocol, a participating server has crashed and then recovered, the server can decide to abort the transaction unilaterally.

One approach to validate a distributed transaction in optimistic concurrency control, is that each of the servers participating in a particular transaction validates its part of the transaction using the same globally unique transaction number at the start of the validation.

Using the one-phase atomic commit protocol, the coordinator communicates the commit or abort request to all of the participants and keeps on repeating the request until all of them have acknowledged that they have carried it out. This allows a participant to make a unilateral decision to abort a transaction when the client requests a commit.

A phantom deadlock could be detected if a waiting transaction in a deadlock cycle aborts during the deadlock detection procedure.

Results for question 4.

What are possible reasons that prevent a transaction server from being able to commit its part of a distributed transaction? Select the correct answer(s).

the failure of validation at the transaction server

writing an object previously written in the same transaction

a crash and subsequent recovery of the transaction server after it has promised to commit the transaction

reading an object previously written in the same transaction

a crash and following replacement of the transaction server during the progress of the distributed transaction

writing an object previously written by a different transaction

the resolution of a deadlock that leads to the aborting of the transaction

Results for question 5.

Which of the following statements about coordination problems in a distributed system are correct? Select correct statement(s)

In uniform reliable multicast, when a node receives a new message, it forwards it to all nodes in the group and then delivers it to the application layer

In uniform reliable multicast, when a node receives a new message, it delivers it to the application layer and then forwards it to all nodes in the group.

Uniform agreement for multicast is the combination of total and causal ordering.

A causal order multicast is also a FIFO order multicast.

A ring-based leader election algorithm has better turnaround time compared to a bully algorithm.

A reliable multicast requires a reliable failure detector and can be implemented only on a synchronous communication network.

In a centralised solution to the distributed mutual-exclusion, requests can be granted in happened-before order by using Lamport clocks and PIDs.

A total order multicast is also a FIFO order multicast.

Results for question 6. **(iff 表示 if and only if)**

What can we know if we use Lamport clocks?

L(a)<L(b) → *a happened before b*

If L(a)= L(b) then a=b

Iff L(a)< L(b) then *a happened before b*

a happened before b → L(a)< L(b)

Results for question 7.

An event *x* can be timestamped in a distributed system with the Lamport clock timestamp denoted below by *L(x)*. What can we conclude by looking at the Lamport timestamps of the two events, *a* and *b*? Select the correct answer(s).

If L(a)<L(b) then a could have caused b.

If L(a)<L(b)then a occurred in real-time before b.

If, and only if, L(a)<L(b)then a happened before b.

If L(a)<L(b)then a happened before b.

If L(a)=L(b)then a=b.

If L(a)<L(b)then a happened after b

If L(a) is at least L(b) then a did not happened before b.

If, but not only if, b happened before b then L(a) <L(b).

Results for question 8.

What is the most that we know if we use vector clocks?

If *a* happened before *b* then V(a)<V(b)

if V(a)=V(b) then *a* and *b* are unordered

V(a)<V(b) if and only if *a* happened before *b*

If V(a)<V(b) then *a* happened before *b*

Results for question 9.

A replicated service is *linearizable* if for any execution there is some interleaving of operations that meets the specification of a non-replicated service, and matches the program order of operations in the real execution.

True

False

Results for question 10.

A replicated service with active replication requires deterministic execution.

True

False

Results for question 11.

What is sent by the primary replica manager to the backup replica manager in a passive replicated system?

A copy of the request time stamped with a vector clock that provides the information on how to order the request.

A unique identifier, the state change and the response.

A snapshot of the state after each request has been handled.

A copy of the original request together with a unique identifier.

Feedback

General Feedback

We need to update the replica but also prepare it for being elected the new primary. It should then have the unique identifier and the reply if the client will resend the request.

Results for question 12.

Assume we have a server that is up with the probability *p* and use *n* replicated servers to increase reliability of a service. What will the probability be that the service is up, assuming that we only need one node to be up in order to provide the service?

$1-(1-p)^n$

$1-p^n$

$(p-1)^n$

$p^n - 1$

Results for question 13.

Can we implement a RPC system with *exactly-once* semantics in an asynchronous system with non-failing nodes but unreliable links?

yes, simply resend the request until an acknowledgment is received

no, we can only achieve at-most-once or at-least-once but not both

no, since messages can be lost a reply is not guaranteed to reach the client

yes, if the client keeps re-sending a uniquely tagged request until a reply is received and a server keeps track of all handled request in order not to duplicate a request

Results for question 14.

If a RPC call with *at least once* semantics fails, we know that:

the call has been executed at least once

the call has either been executed once or not at all

the call has not been executed at all

the call might have been executed at least once

Feedback

General Feedback

If the call fails it could still be that the request was sent multiple times.

Results for question 15.

Which of the following statements about remote invocation techniques for communication in distributed systems are correct? Select correct statement(s)

Even if a failure has been reported, the caller knows that the remote procedure with *at-least-once* RPC semantics has been executed at least once.

On receiving an object reference as an input/output parameter or a return result in a remote method invocation, a process can then access this object using remote method invocation instead of transmitting the object value across the network.

A non-idempotent operation is an operation that can be performed repeatedly with the same effect as if it had been performed exactly once.

Unlike a regular procedure call, a call to a *void* remote procedure returns as soon as the call request is sent to the destination computer where the procedure is to be executed.

With *at-most-once* semantics, the caller receives either a result, in which case the caller knows that the procedure was executed exactly once, or an exception informing it that no result was received, in which case the procedure will have been executed either once or not at all.

Each remote procedure call is executed in a separate process (thread) on the server side.

Results for question 16.

Unlike a normal remote procedure call, a call to a void remote procedure returns as soon as the call request is sent to the destination computer where the procedure is to be executed.

True

False

Feedback

General Feedback

A void remote procedure call is synchronous (blocking) like a normal procedure call and it returns as soon as the procedure is either successfully completed or an error occurs.

Results for question 17.

According to the CAP theorem, you can have a consistent and always available system even if you're in an environment where you face network partitions.

True

False

Results for question 18.

In a distributed hash table of *n* nodes, with log(n)links in the routing finder table of each node, the maximum number of hops in any route is

O(n)

$O(n^2)$

O(log (n))

O(n log n)

Results for question 19.

Communication using UDP sockets is connectionless and stream-based.

True

False

Results for question 20.

What is not provided by TCP?

flow control, not to overflow the receive buffer

a full-duplex stream between two processes

congestion control, to avoid network congestion

a guaranteed delivery of messages

Results for question 21.

Which of the following statements about characteristics and properties of distributed systems are correct? Select correct statement(s).

Replication transparency in a distributed system enables processes have knowledge of a replication scheme and can take advantage of it.

Access transparency in a distributed system enables remote resources to be accessed using location independent names.

Concurrency transparency in a distributed system enables several processes to operate concurrently using shared resources without interference between them.

Failure transparency in a distributed system allows users and applications to complete their tasks despite the failure of system components.

In an asynchronous distributed system, there are no upper bounds on the execution times of processes and on message transmission delays; however, lower bounds on transmission delays can be known (measured or estimated).

The major aspect that makes distributed systems different from non-distributed systems is privacy and authentication.

A perfect failure detector in an asynchronous distributed system can be implemented using heartbeats between processes.

In a synchronous distributed system, the time to receive a message transmitted over a channel has known upper bound; however, lower and upper bounds on the time to execute each step of a process is unknown.

In a client-server architecture, the server is a proactive process whereas the client is a reactive process.

In a peer-to-peer architecture all processes play similar roles, interacting cooperatively without any distinction between client and server processes.

Results for question 22.

What is the most significant difference between a synchronous and an asynchronous distributed system?

the ability to synchronize clocks

how concurrency is implemented

possibility of perfect failure detectors

location transparent operations

Results for question 23.

What is meant by ***time uncoupling*** in a communication framework?

A sender does not need to know the name or identifier of the receiver.

The sender does not wait for a acknowledgment from the receiver.

Messages are resent if lost, providing a reliable service.

Sender and receiver need not be active at the same time.

Results for question 24.

Can a synchronous communication interface be used in an asynchronous way?

no, synchronizing clocks will always take time

no, the send operation will block and stop the whole execution

yes, by minimizing the latency for the receive operation

yes, by performing the send operation in a separate thread

Results for question 25.

What is meant by *causal ordering* in a communication framework?

If a sender is sending two messages m1 and m2, then a receiver will be delivered m1 before m2.

If two clients send messages m1 and m2 independently of each other, then a receiver will receive these in the real-time order in which the send operations were issued.

Clients will be able to send messages in a *round-robin* order thus preserving inter-ordering relationships.

If a client is delivered two messages m1 and m2, then m1 could not have been sent by someone after having being delivered m2.

Feedback

General Feedback

If a process sees m2 and then sends m1 we have a potential causal relationship that we should preserve.

Results for question 26.

Which of the following statements about a global state of a distributed system are correct? Select correct statement(s).

A cut of the system's execution is a subset of its global history, i.e., a union of prefixes of process histories.

The predicates associated with the state of an object being garbage, the system being deadlocked, or the system being terminated are all *unstable predicates.*

The snapshot algorithm of Chandy and Lamport assumes that neither channels nor processes fail – communication is reliable so that every message sent is eventually received intact, exactly once.

A finite prefix of the process's history is a sequence of all events which causes state transitions of the process.

Solving the distributed termination detection problem by observing a global state requires testing whether each process has halted, i.e., all processes are passive (or idle).

Suppose two successive events in a linearization are receiving messages by two processes. In that case, one may swap the order of these two events within the linearization and derive a run that still passes through only consistent global states.

A *linearization* or *consistent run* is an ordering of the events in global history that is consistent with this happened-before relation on the global history.

Results for question 27.

What is the definition of a consistent cut?

if ***e*** and ***f*** are in the cut then ***e*** and ***f*** have a causal order

if ***e*** is in the cut and ***f*** happened-before ***e*** then ***f*** is in the cut

if ***e*** happened before ***f*** and ***e*** is in the cut then ***f*** is in the cut

all events in the cut are strictly ordered in a *happened before* order

Results for question 28.

Why is the notion of consistent cut important?

it defines a state that possibly occurred during an execution

it defines the difference between synchronous and asynchronous systems it defines a state that did occur during an execution

used to resolve two-phase locking

Results for question 29.

Which of the following statements about a predicate of the global state of a distributed system are correct? Select the correct statement(s).

If there is a consistent global state through which a linearization of the execution history passes such that a global state predicate is True, then the predicate was ***definitely*** true in the execution.

An non-stable global state predicate could hold true in a consistent state but then be false in future states.

If a system enters a consistent global state where the stable global state predicate is true, the predicate will remain true in all future states.

A non-stable global state predicate will never be true in any consistent state reachable from the original state.

A stable global state predicate is true in all consistent global states.

If there is a consistent global state through which a linearization of the execution history passes such that a global state predicate is True, then the predicate was ***possibly*** true in the execution.

Results for question 30.

How does a distributed file server, e.g. NSF server, know at what position to read and write to?

each read and write operation holds the position

not needed since all read and write operations are on a whole file

NFS only allow read operations and therefore does not need position information it keeps a file table entry with a read/write position

Results for question 31.

What is the advantage of a bully algorithm compared to a ring based algorithm?

two nodes will never be elected even if we have an unreliable failuredetector

nodes can easily change priority in-between elections

better turnaround time

better worst case number of messages

Results for question 32.

When does Ricart and Agrawalas mutual exclusion algorithm perform better than a central solution?

under low congestion when hardly any conflict will occur

when the number of nodes is four or less

under high congestion since only one message is needed to release and obtain the lock

when there is a risk of nodes crashing

Results for question 33.

How are FIFO, causal and total order multicast related?

a FIFO order is also a total order

a total order is also a FIFO order

a causal order is also a total order

a causal order is also a FIFO order

Results for question 34.

What accuracy can be provided using Christian's algorithm?

$\pm(T_{round} \div 2)$

$\pm(T_{round} \div 2 - \text{min latency})$

$\pm(T_{round} \times 2 - \text{min latency})$

$\pm(T_{round})$

Results for question 35.

At time 114 a node receives a NTP reply with the following information: request sent at 79, received at 108, reply sent at 115. How should it adjust its time?

15

Feedback

General Feedback

Section 14.3.4

You calculate the offset by $[(T2-T1)+(T3-T4)]/2$. It gives you an offset of +15.