ID2201 Distributed Systems

2015-10-30

Answers to the questions:

Section A

A:1 1p. What aspect makes distributed systems most different from non-distributed systems?

harder to handle failure

Comment: Failures are harder to detect accurately, especially in an asynchronous system. Concurrency is a problem even in non-distributed systems.

A:2 1p. What is meant by location transparency?

remote resources are accessed using location independent names

Comment: Section 1.5.7

A:3 1p. What is provided by TCP?

a full-duplex stream between two processes

Comment: Section 3.4.6

A:4 1p. What is a good reason for choosing UDP rather than TCP?

you have small messages that should be sent with little delay

Comment: Section 3.4

A:5 1p. What is meant by *time uncoupling* in a communication framework?

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

Comment:

A:6 1p. What is meant by *causal ordering* in a communication framework?

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

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

A:7 1p. If a RPC call with *at least once* semantics fails, we know that:

the call might have been executed at least once

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

A:8 1p. Does Erlang provide a form of location transparency?

Yes - a process can use a process identifier without having to know the address

of the node where the process lives.

Comment:

A:9 1p. What accuracy can be provided using Christian's algorithm?

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

Comment: Section 11.3.2

A:10 1p. What does the reply from a NTP server contain?

send and receive time of request and send time of reply

Comment: Section 11.3.4

A:11 1p. What is true for events A and B?

if A caused B then A *happened before* B

Comment: Section

A:12 1p. What can we know if we use Lamport clocks?

if $L(a) < L(b)$ then a could have caused b

Comment: Section 11.4

A:13 1p. When is it problematic to use vectors clocks?

when we have a dynamic set of processes

Comment: Section

A:14 1p. What is the definition of a consistent cut?

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

Comment: Section 11.5

A:15 1p. What is the definition of a stable global state predicate?

if a system enters a state where the predicate holds true it will remain true in all future states

Comment: Section 11.5.2

A:16 1p. What is the benefit of a reliable multicast?

messages are guaranteed to be delivered to all correct processes

Comment: Section 12.4.2

A:17 1p. What is the difference between *uniform agreement* and *non-uniform agreement* for multicast?

Uniform agreement includes the behaviour of non-correct node.

Comment:

A:18 1p. What does Lamport clocks give us when implementing distributed

mutual-exclusion?

request are granted in *happened before* order

Comment: Section 12.2

A:19 1p. What is two-phase locking in a transaction?

not taking any locks once a lock has been released

Comment: Section 13.4

A:20 1p. What does it mean that a transaction meets the isolation property?

intermediate results must not be visible to other transactions

Comment: Section 13.2

A:21 1p. In a distributed transaction one often use two-phase commit, why is this better than one-phase commit?

We ensure that if one transaction aborts then no transaction will commit.

Comment: Using one-phase commit we can order the servers to commit but any of the serveres could say "sorry, I had to abort".

A:22 1p. In a view-synchronous group membership protocol, a process that enters the group, and is included in the delivered view, will be guaranteed to be delivered:

all messages starting from the view where it is included in the group

Comment: It could take some time between we issue a request to join and the time when we're included in the view. From this point in time we should see all messages.

A:23 1p. What is sent by the primary replica manager to the backup replica manager in a passive replicated system?

A unique identifier, the state change and the response.

Comment: 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.

A:24 1p. What is the purpose of hashing in a DHT?

to generate uniformly distributed keys

Comment:

Section B

B:1 2p. When defining a request message layer, one has the option of choosing to provide *at-most-once* or *at-least-once* semantics. Describe an advantage with the *at-least-once* semantics but also describe what it means to the appli-

cation layer.

Answer: In an *at-least-once* layer the implementation is free to resend messages if they are possibly lost. This will be able to hide some network problems and thus provide a more reliable service. The application layer needs to adapt to the situation where a request could have been duplicated by the messaging layer. This can be done by only using idempotent operations.

B:2 2p. Briefly describe a situation that meets the requirements of total order multicast but not to FIFO order multicast.

Answer: Two processes, A and B, multicast two messages each, a_1, a_2 and b_1 and b_2 and these are delivered in the order a_2, a_1, b_1 and b_2 by both nodes. Both nodes see the same order, i.e. total-order, but the messages from A were not delivered in FIFO order.

B:3 2p. If you need to replicate a server that mainly handles compute intensive read request i.e. request that do not change the state of the server, what would then be the best strategy for replication, active or passive, and why?

Answer: If the request is compute intensive it would be better to use a passive replicated system. Then only the primary will do the computation and since it is a read request the message to the passive replicas need not include a state change.

In an active replication scheme that multicast all request all machines would be heavily loaded. If the front-end can determine that it is a read request it could possibly send this to only one replica thus implementing load balancing. We could then improve the throughput of the system but it requires that the front-end can determine the type of request (which might not be possible).

B:4 2p. Describe an efficient algorithm that allows you to detect a distributed dead-lock.

Answer: When suspended on a lock, send a token forward to the porcess that holds the lock. The token is forwarded by any process that is suspended waiting for a lock. If the token finds its way back to the initial process then we have a dead-lock.

Taking a snap-shot of the whole system does of course work but can not be regarded as efficient.

B:5 2p. In a quorum based algorithm the quorum is often chosen to be the majority of processes. You could however choose other quorums, what is the most important criteria when you divide nodes into quorums?

Answer: That any two quorums overlap in at least one node so that it prevents two quorums to be formed.

B:6 2p. Describe with a message diagram the difference between a service that is *linearizable* and one that only provides *sequential consistency*. Why would we settle for an service that only gives us sequential consistency?

Answer: A linearizable service respects real-time order; if one node performs an operation in real-time before another node performs an operation, then these

operations must be executed in this order.

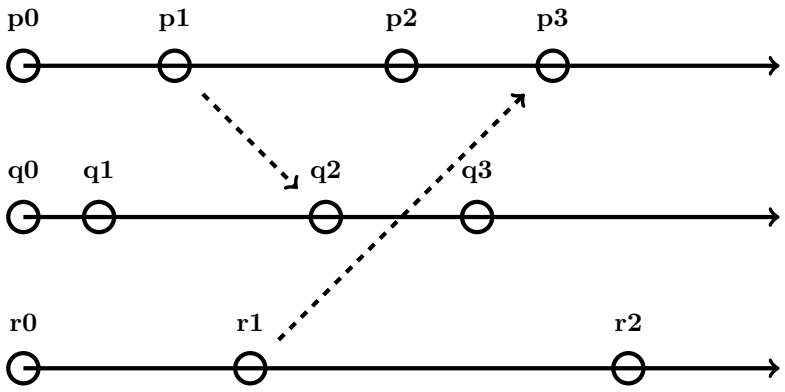
Sequential consistency is easier to implement since we only have to respect the program order (we still have to maintain a sequential order of all operations).

B:7 2p. Why would you use a snap-shot algorithm, what is collected and what can you determine?

Answer: The snap-shot will collect a consistent global state. It does so by collecting local states and all messages that are in transit i.e. that have been sent but not yet arrived.

If one can prove that a stable predicate holds in the generated global state then it is also true in the global state of the real execution.

B:8 2p. Draw two lines, one that represents a *consistens cut* and one that represents an *inconsistent cut*.



Answer: The consistent cut should not include any receive event if the send event is not included. The opposite holds for the inconsistent cut.

Section C

C:1 4p. What we are often looking for is what caused something to happen in a distributed system. To our help we have Lamport clocks, vector clocks and real-time clocks. How are these clocks related to each other and how are they related to causality? Describe the pros and cons of using these clocks, and how good they are in helping us track down the events that caused something to happen.

Answer: If we have no knowledge of the application layer the best thing we can do is to track *happened-before* order. The vector clocks would give us a perfect picture of the happened-before order and all events that could possibly

have been the cause.

A Lamport clock will give us a set of events that is larger i.e. we will have events in the set that did not happened-before.

If we have perfect synchronized clocks the real-time clock will give us all events that are in the happen-before set. It will also give us events that happened real-time before but not necessarily happen-before.

If the real-time clocks are out of sync we might fail to include all events that are in the happend-before set and thus fail to find all events that are in causal realationship.

Vector clocks will take up space and are complex to manage, especially in dynamic systems.

C:2 4p. Assume that we have a distributed system where processes communicate by sending messages. We have an *unstable predicate* that we want to determine if it is true during an execution. How would we go about to answer this question? Outline what an implementation would look like and what limitations there would be. What could you do to limit the amount of messages and the size of messages?

Answer: We could use vector clocks and let each process send its state, tagged with a vector time-stamp, to a monitor process. The monitor would group the states into possible *global consistent states*. These states are then ordered forming a latices. If the predicate that we are interested in is true for some nodes in this lattice so that it is impossible to find a path from the original state to the final state without going through a true state then the predicate was definitely true during the execution.

In order to limit the number of messages and the size of the messages, the processes need only send in a state when the parameters of the predicate change. Moreover they need only send in the information that is relevant to determine the predicate. This will of course only work if the predicate is known aforehand.

C:3 4p. You're building a high performance transaction server using time-stamp concurrency control. Describe what information you save for each object in the data-base and the rules for the read and write operations.

When will a transaction suspend and when will it have to abort? Why would it be an advantage to save multiple committed values and how would you make sure that you're not storing data that is no longer needed?

Answer: A short description that shows how tentative write operations are tied and how successful read operations are recorded. Transactions will abort if they issue a read operation or write operation that arrives "too late". It will only suspend if it tries to read an uncommitted value.

If one would allow a sequence of committed values one would allow more read operations to be performed even if they arrive late. We could keep track of the transaction with the lowest time-stamp and always truncate sequences so that earlier values are removed. This could be done by each transaction or by a separate daemon that would go through all data items.