Contributors

Menelaos Kokolios

Distributed Systems - May 2013 Exam

CONTRIBUTION APPRECIATED!

1. (a) List three good reasons to build a distributed system. [*2 marks*]

   (b) Why is a proxy server more appropriate for a search engine than a social network? [*2 marks*]

   (c) A synchronous system has known bounds on three properties that an asynchronous system does not.

      i. What three properties are they? [*3 marks*]

      ii. Any asynchronous system can be made synchronous simply by assuming very large bounds which will not realistically be broken. Why is this not commonly done? [*2 marks*]

   (d) Two internet protocols, UDP and TCP provide message transmission with differing failure semantics. Which of the two is used by HTTP for sending documents (web pages) from server to client? Explain why. [*4 marks*]

   (e) Explain the difference between a process omission failure and a process arbitrary failure and give an example of both. [*4 marks*]

   (f) You and a friend would like to meet up in a bar for lunch and are arranging this via text messaging. Neither of you wishes to turn up alone. Assuming that there is some possibility that a text message is dropped, devise a protocol that ensures that either both or neither of you turn up to lunch, or explain why no such protocol exists. [*3 marks*]

   (g) My mother wishes to send me a secure message, so she uses my very secure 2048 bit public key to encrypt the message. The message though is simply "Hi son".

      i. What can go wrong? [*2 marks*]

      ii. What, if anything, should I do upon receiving this message to ensure that further communication is secure? [*2 marks*]

      iii. Assuming my mother has a public encryption key such that two-way communication is possible, why might I send her a (possibly newly generated) private-shared key? [*1 mark*]

Q1.

(a)

Good reasons to build a distributed system:

- Tasks get done faster
- Can be made more resilient - if one computer fails, another takes over

- Load balancing and source sharing
- Sometimes systems are inherently distributed - people from different locations collaborate on tasks
- Brings out many natural questions about how natural world ecosystems, economies, emergent behaviours work

(all the reasons were taken from the slides...)

(b)

Did we do anything this year about proxies? I don't think so, please correct me.

(c)

i. A synchronous system has known to have bounds on:

- Transmission (message delivery) time, say m.
- Execution and computation time, say c.
- The clock's drift rate from the real time value is bounded by a known value.

=> Round duration time m+c.

Also the clocks have to be synchronised for synchronous communication.

I don't think it's valid - synchronous system doesn't assume any clock synchronisation. We operate with time changes rather than global time.

ii. Very large bounds on asynchronous systems are not commonly used to make the system synchronous, because such bounds are unrealistic and because in general asynchronous distributed systems are more abstract and if they work on one system they are also likely to work on another one.

This is all I found in Alan Clark's slides, nothing in this year's.

Another answer would be that the data sent in round x can only be used in round x + 1 as synchronous systems tend to be round based. Assuming large bounds would make the system really really slow.

(d)

TCP is the one used for HTTP requests, as it is connection based and used for larger requests. Also, UDP suffers from possible omission failures, does not provide error correction, while TCP provides error detection + correction and guaranteed message delivery service.

(e) From EXC: Omission failure occurs when a process fails to send a message. Arbitrary failure occurs when a process sends a wrong message at a wrong time.

(f)

There exists no such protocol via text messaging to ensure that neither of us turns up alone. Even if we agree to send acknowledgment (confirmation) we need to take into account that the confirmation may fail as well. This is the two generals' problem.

(g)

i.

- Eve can eavesdrop and replay the message later. :)
- The transmission can be intercepted and the message can be corrupted - changed to garbage or modified in favour of Mallory.

ii.

You should make up an encryption key K_mom_you, add a nonce and encrypt it with K_mom_public and send it back. I cannot understand how this prevents Mallory from intercepting the message again.? If Mallory intercepts again, she will not be able to decrypt the message because she doesn't know K_mom_private, thus she wouldn't be able to find out K_mom_you.

use time values- like in kerberos

iii.

In order to set up a speedier secret-key communication. +1

symmetric encryption

for sending sequence of messages

2. (a) A physical clock H is said to be correct with respect to a given bound p if, for two real times t and t', such that $t < t'$, we have: $(1-p)(t'-t) \leq H(t) < H(t') \leq (1+p)(t'-t)$. What important feature does this definition have and how does it affect what we do when we determine a clock to be running fast? [*4 marks*]

(b) Suggest a way that you can synchronise clocks with a friend using only mobile phone text messaging. You do not have any bounds on how long it takes a message to be sent and received, nor how quickly either of you can respond to messages. Give a bound on how well your clocks are synchronised. [*4 marks*]

(c) Recall from the course Lamport logical clocks and Vector clocks.

i. We are using Lamport logical clocks. Two events, e_1 and e_2 are stamped such that: $L(e_1) = L(e_2)$, what can we say about the two events e_1 and e_2? [*2 marks*]

ii. Lamport clocks give a partial ordering to a set of events in a distributed system. One can enforce a total ordering by using the process identifier at each process. A colleague suggests we do the same thing for Vector clocks. What do you think of this idea? Explain your reasoning. [*4 marks*]

(d) A Snapshot algorithm aims to record a global state of a distributed system using local states recorded at separate times. For each of these applications explain whether a global snapshot algorithm would be appropriate:

i. A distributed finance application stores information about customers and their accounts. Such information may be deleted once the customer deletes their account and any transactions involving the customer have been fully completed. [*2 marks*]

ii. The same scenario but the user account may be reactivated within 30 days of deletion. [*1 mark*]

iii. The same distributed finance application wishes to know if two accounts located separately were ever simultaneously negative. [*2 marks*]

iv. The same distributed finance application wishes to know if either of the two accounts have ever been negative. [*2 marks*]

(e) The distributed debugging algorithm that we looked at defined two relations *definitely(p)* and *possibly(p)*. For each of the following implications explain whether or not it holds:

i. $\neg possibly(p) \Rightarrow definitely(\neg p)$ [*2 marks*]

ii. $definitely(p) \Rightarrow \neg possibly(\neg p)$ [*2 marks*]

Q2.

(a) The important feature of this definition is monotonicity, meaning that if `t<t'`, then `H(t) < H(t')`. If a clock is running fast, we would break monotonicity if we were to set it back. Instead, we represent the time as `C(t) = α H(t) + β`. By decreasing beta, we retain `C(t) < C(t')` for `t<t'`.

(b) I propose Christian's method. Suppose I send a message requesting synchronization to my friend at time T_{sent}, he receives it at time t, and sends back a response, which I receive at time T_{recv}. The round trip time is $T_{round} = T_{recv} - T_{sent}$. Therefore, I would set my clock to $t + T_{round}/2$,

assuming it takes approximately the same time for both messages to be delivered. Suppose min is the minimum time needed for a message to arrive. Therefore, the time on my friends watch at the time when I receive his reply is at least t + min, or at most t + T_{round} - min. The interval is therefore T_{round} - 2min, meaning that the accuracy is T_{round} /2 + min.
(what about the time for respond the question mentioned?)

(c)
i. L(e1) = L(e2), the two events e1 & e2 are concurrent.
ii. That would not be suitable because if we were to break ties with process identifiers the property of vector clocks that V(e1) < V(e2) implies e1 -> e2 would no longer be valid.

(d)
i. A distributed finance application stores information about customers and their accounts. Such information may be deleted once the customer deletes their account and any transactions involving the customer have been fully completed.
The snapshot algorithm is useful for detecting stable properties. Here, once the account is deleted, it is deleted afterwards. Same holds for finished transactions. So, if someone takes snapshots of the system and checks for these conditions to hold, it is enough to see the conditions hold in a snapshot. In my opinion, it works here.

ii. The same scenario but the user account may be reactivated within 30 days of deletion.

Don't understand what we should do here, though. I would take a snapshot and check modification date of the client account. If it was deleted later than 30 days ago, I would check if all transactions are finished. If so, we are free to remove the account.

iii. The same distributed finance application wishes to know if two accounts located separately were ever simultaneously negative.

Not sure: If it's just one message required to transfer money from one account to the other, e.g. acc1 sends amount to acc2 like so:

1. acc1.balance -= amount;
2. acc1.sendMsg(acc2, amount);
3. acc2.recvMsg(amount) { acc2.balance += amount};

then it's a proper algorithm to do so (because we also capture the channel states). However, if we are expecting some multi-message exchange, then no… ? (I'm supporting myself with the example given during the lecture - we may just capture reaction to some message - one that we don't capture).

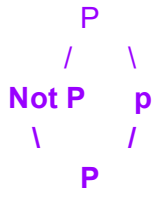
iv. The same distributed finance application wishes to know if either of the two accounts have ever been negative.

The answer would be possibly true - not definitely. Again, not sure.

(e)
i. ¬possibly(p) ⇒ definitely(¬p)
i. Holds. If p is impossible then ¬p definitely holds.
ii. definitely(p) ⇒ ¬possibly(¬p)
ii. ***DOESN'T HOLD****. definitely(p) means that we have at least one state in each linearization that p is true.* ***¬possibly(¬p)*** *means that for all states p is true, so we cannot infer the latter from the former (the 1st is a subset of the second).***WTF? WTF+1+1** This makes sense to me ^^ WTF - 1

"The statement definitely(p) means that for all linearization L of H, there is a consistent global state S through which L passes such that S(p) is true"
"The statement possibly(p) means that there is a consistent global state S through which at least one linearization of H passes such that S(p) is true." /from slides/

Consider this possible history



Then definitely(P) since the first state had P, but also possibly(not P) since one history has not P holding.

Q3.

3. (a) This question concerns data marshalling

   i. What is data marshalling and why is it necessary? [*2 marks*]

   ii. Some data marshalling techniques send the type of data objects together with the data objects themselves as a part of the marshalled data stream, others do not. For both approaches describe one advantage and the general situation in which it would be used. [*4 marks*]

   (b) It has been measured that if a failure detector process P sends an "are-you-alive" message to process Q it can expect a response from a live process Q within time t in 90% of cases. If P sends Q an "are-you-alive" message and does not receive a response within time t what can we say about the probability that the process Q has failed? [*2 marks*]

   (c) In the course we considered four distributed mutual exclusion algorithms. We analysed the four in terms of their performance:

   i. In the Ring Based algorithm how many messages does it take to enter the critical section? [*2 marks*]

   ii. What is the disadvantage in bandwidth terms of this algorithm and in what situation would it therefore be most applicable? [*2 marks*]

   (d) This question concerns building a multicast layer on top of a unicast messaging system:

   i. A simple Basic Multicast can be implemented by the sending process sending to all processes in a loop using a $send(p, m)$ operation. If we assume that the $send(p, m)$ operation is a reliable one-to-one operation what property does this basic multicast not have and why? [*2 marks*]

   ii. Suppose you are using a sequencer to provide a total order on all broadcast messages. Why might one wish senders to additionally stamp their messages (to the sequencer) with their Lamport stamps? [*3 marks*]

   (e) For a synchronous system it is known that we cannot provide an algorithm guaranteed to reach consensus if the number of processes which fail is greater than or equal to the total number of processes divided by three. What, if any, are the equivalent conditions for an asynchronous system? [*2 marks*]

   (f) This question concerns distributed systems and operating system concepts:

   i. What are 'threads'? [*2 marks*]

   ii. Give two advantages of threads over separate processes which a server process may utilise? [*2 marks*]

   iii. How might a client-process make use of threads? [*2 marks*]

(a) I don't think we covered data marshalling this year.

We have covered it briefly in the distributed objects lecture.

i) It takes objects and serialises them into XML for transfer over networks. This allows different internal encodings to talk.

ii) (Definitely not covered this year)

(b)

We have two events:

a - process fails;

b - process doesn't respond.

From Bayes rules P(a|b) = \frac {P(b|a)P(a)}{P(b)} and we know that process will not respond, if it failed -> P(b|a) = 1.

Thus, $P(a|b) = \frac{P(a)}{P(b)}$. We also know P(b)=100%-90%=10%.

Anyone has some idea how to finish it?

Or maybe the correct answer now is that, we need to have some statistics on how faulty the processes are (P(a)), and our final probability is P(a|b) = 0.10 * P(a).

Should be correct, since in the other side we have written - "We can learn the probability distribution of message delivery time, and accordingly estimate the probability of failure"

This is an example of the false-positive paradox, i.e. what we learn depends on P(a).

(c)

i. between 0 and N -1

ii. Even if processes do not require access to the critical section, the algorithm continues to consume bandwidth.Situation when all (most) processes need short frequent access to the critical section.

(d)

i. Agreement - messages not delivered in the same order at all processes ??

Answer 2: Agreement: Basic multicast is assuming that node is not failed. If node is failed, so the agreement will be violated because some node will not deliver the message.

ii. Process p sends a request for a sequence number to the sequencer before q does; however, q's request arrives first and thus receives priority over p's request. A Lamport timestamp would preserve the happened-before relation in regards to which request was sent first to the sequencer.

(e) No such condition for asynchronous system; consensus not guaranteed.

(f)

i. Threads are processes inside a process ie. they share the same memory space, hence the communication is easier.

ii. Threads have the advantage that they have access to the same memory space, thus they can communicate between themselves more easily. Also, threads need more or less the same information as the process itself, hence switching execution between threads is less work for the OS.

iii.

Client-processes can make use of threads to execute concurrently (that is one for each client), hence if client A is slow, client B does not have to wait for A to finish, but can continue working on a separate thread.

Thank you for your time.

Best of luck! Oh stop it you :)