CSCI/ECEN 5673: Distributed Systems
Spring 2017
Homework 2
Due Date: 02/28/2017

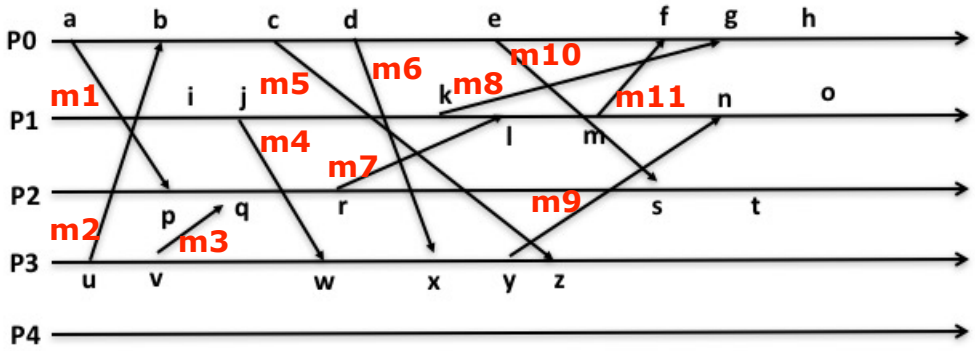
Please submit a hardcopy of your answers in class on Tuesday, February 28, 2017. BBA students may submit a PDF copy of their answers via the submission link provided on Moodle. Write your answers in the space provided. Please DO NOT use any extra space. The space provided is sufficient for answering the questions.

Topics covered: Message ordering, Consensus.
Lecture Sets: three and four.

Honor Code Pledge: On my honor, as a University of Colorado at Boulder student, I have neither given nor received unauthorized assistance on this work.

Name: Ishan Shah

1. Consider the following figure from Homework 1 that shows five processes ***(P0, P1, P2, P3, P4)*** with events ***a, b, c,*** ... and messages communicating between them.



a) **[3 Points]** Identify two messages between different pairs of processes, i.e. m1 between Pi, Pj and m2 between Pn, Pm, such that Pi ≠ Pj ≠ Pn ≠ Pm and m1 and m2 are causally related.

Pi=P0, Pj=P3, m1=c-z
Pn=P0, Pm=P3, m2=d-x

send(m1) -> send(m2) is true because c and d are on the same process.

b) **[3 Points]** Is there a violation of FIFO ordering between messages in this figure? If yes, provide one example of the corresponding messages.

Yes, there is a violation of FIFO ordering.

m1=c-z and m2=d-x
c happened before d on P0, but m1 was delivered after m2

c) **[3 Points]** Is there a violation of causal ordering between messages in this figure? If yes, provide one example of the corresponding messages. Your example should not include the messages where there is a violation of FIFO ordering.

Apart from the examples of FIFO ordering violation, there is no Causal ordering violation.

d) **[6 Points]** Provide a total ordering of messages that preserves causal ordering in this figure. Is your total ordering unique? If not, provide a different total ordering of messages that preserves causal ordering.

Total order - m1, m2, m3, m4, m5, m6, m8, m7, m10, m9, m11

No, total order is not unique

Different total order - m1, m2, m3, m4, m5, m6, m8, m7, m10, m11, m9

2. **[8 Points]** Provide an example distributed application for each of the following scenarios (total four examples). Explain your answers.

a) Scenario one: No specific message delivery ordering is needed.

Any courier service like UPS/FedEx software. They just want to deliver all packages as fast as they can. Whatever the packages are in their truck, they create a route so that fastest delivery happens and not an ordered one.

b) Scenario two: FIFO message delivery ordering is needed.

Text messaging application where no groups are there but only one to one texting is available. In this case every text from a sender should arrive in a FIFO manner. But, overall ordering amongst all processes is not necessary.

c) Scenario three: Causal message delivery ordering is needed, but not total ordering.

Chat rooms with group chats. In this case every user needs to see all the messages in the causal order to be able to maintain contextual flow and understanding by knowing everything the sender knew/received.

d) Scenario four: Total message delivery ordering is needed.

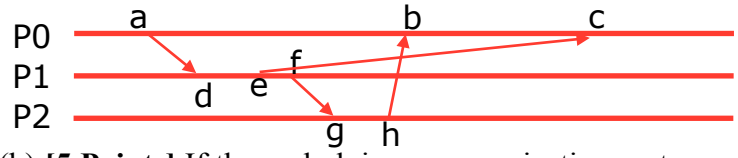
Bank software. A withdrawal might be rejected even after funds were deposited before the withdrawal request was made if there was just causal ordering in place.

3. Consider the following definition of causal ordering among message receive events: if *send* (*m1*) → *send*(*m2*) and *m1* and *m2* are received by the same process, then *m1* must be received before *m2*.

   Suppose an underlying communication system provides reliable, FIFO message communication.

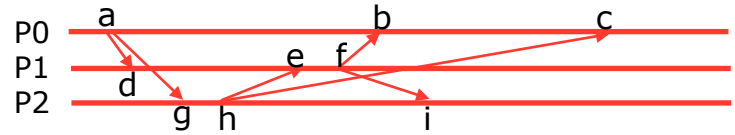
   (a) **[5 Points]** If the underlying communication system provides only unicast message exchanges, does it preserve causal ordering among message receive events? If yes, provide supporting arguments, if no, provide a counter example.

No, it doesn't preserve causal ordering among receive events. As we can see, e->f, f->g, g->h implies e->h but b->c is true which means that sender events e and h have e->h relation but corresponding receive events did not preserve that order and violated the causal ordering but did not violate FIFO



   (b) **[5 Points]** If the underlying communication system provides multicast, does it preserve causal ordering among message receive events? If yes, provide supporting arguments, if no, provide a counter example.

No, it doesn't preserve causal ordering among receive events. As we can see, h->e and e->f implies h->f but b->c is true which means that sender events h and f have e->h relation but corresponding receive events did not preserve that order and violated the causal ordering but did not violate FIFO



4. **[5 Points]** Consider the consensus algorithm for synchronous distributed systems under scenario 3 that we discussed in class. Construct an example to show that the processes may not reach a consensus with only *f* rounds with *n* processes, $n > f$.

For n>f, if there's 1 process failing with each round then for f failures, there has to be one such extra round that has no failures which makes sure that every non-faulty process knows about rest of the participating processes.
For example, if n=3 and f=1 then,
1st round - P2 and P3 multicasts to everyone but P1 sends a message to P2 and then fails i.e. P3 doesn't know about P1
2nd round - P2 and P3 multicasts which means P3 updates its vector for P1 using P2's vector
Hence, 2nd(i.e f+1) round is required if P1 process fails in order for P3 process to update its vector using P2's vector to reflect P1's information

5. **[10 Points]** One limitation of the ISIS ABCAST protocol is that the sequencer node is a performance bottleneck. We can address this by having the role of sequencer rotate among all group members. Provide an algorithm for how you can rotate the sequencer role. Be specific in terms of what messages are used and the relevant content of those messages.

ISIS ABCAST protocol assigns the sequencer role to the process with lowest process ID or number. Which results into a bottleneck or a single point of failure. In order to avoid such bottleneck, a protocol can be designed where we can rotate the role of sequencer amongst all participating processes. Following algorithm can address this issue,

n = number of participating processes
m = number of messages after which the sequencer role is transferred

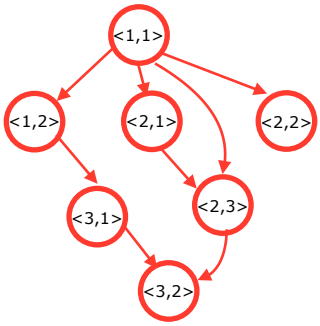
Algorithm - After m messages are sequenced by a sequencer process i.e. it multicasted the total order/sequencing messages for m messages, it will compute which process did the least amount of communication i.e. sent least amount of messages, the current sequencer will send a request to that process for becoming the next sequencer since that process doesn't have heavy workload. The current sequencer will wait for the process to accept the role and then the new process will act as sequencer for the next m messages. Next sequencer first will make sure that m messages are delivered and there's no need for retransmission. In this algorithm, current sequencer will pass on the current sequence number and a newly computed m to the next sequencer.

Now, m should be high enough so that the role changing doesn't become a performance overhead but it shouldn't be too small that a slower process who gets the sequencer role slows down the system or a process gets too much burden. m can be computed empirically using run-time statistical models and should be computed dynamically using the traffic in a particular system.

6. **[10 Points]** Psync provides causal delivery of messages exchanged with in a group of processes. In class it was mentioned that you can construct a total order of message delivery from the context graph by using topological sort. Explain how this can be done. Your algorithm should not use any extra messages.

In order to achieve total order, each process must do an exact same topological sort to compute the same order. The main concern is that future messages sent to the process will invalidate the total ordering. So, the process has to wait for a certain amount of time before it computes an order for that particular layer/level of the graph.

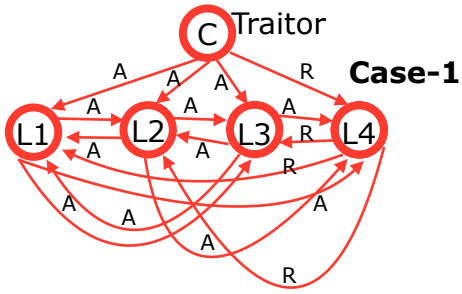
Algorithm - When a message is delivered i.e. every message from the attached message ids with that message is there in the context graph, the process will only attempt to generate a total order till the layer which is not a direct parent of the last delivered message. Each process will do a same sort by using sender ids. If in the same layer of the graph, there are multiple messages, then the sender id will be used to sort. Since, <sender id, seq> will always be unique, it will not create any problems.

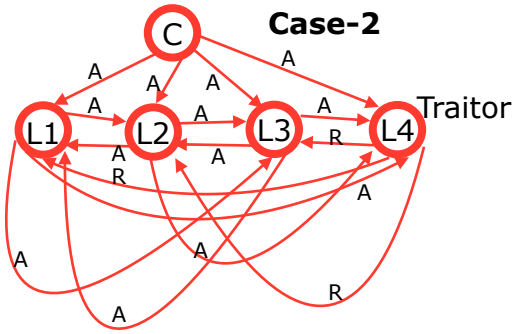


In this case, if the last delivered message was <3,2> then the total order can only be computed till second level. Now, the order will be
<1,1> -> <1,2> -> <2,1> -> <2,2>

here, <1,1> was selected first because it had lower sender id and <2,1> and <2,2> were sorted using seq number.

7. **[12 Points]** The Byzantine Generals algorithm helps reach a consensus when less than one third of the processes undergo byzantine failure. However, it does not suggest how to diagnose such failures. Consider a system of five processes, of which at most one process can be faulty. Examine if the faulty process can be identified without any ambiguity. Investigate all relevant cases.





In both cases, L1, L2 and L3 would get 3 As and 1 R which would result in a consensus. In the the case of retreat also, they would be able to decide using majority consensus. But, we cannot identify a faulty process because in both cases L4 communicates in the same manner.

8. Consider a distributed system comprised on *n* processes that is timed asynchronous, has reliable message delivery, and up to *f* processes may fail (fail stop failure).

   a) **[4 Points]** Does the consensus algorithm we discussed for synchronous distributed system for scenario 3 work in this new scenario? Explain your answer.

   No, it can't work. In a timed asynchronous system, receiver has a statistical approximation of the delay. If the message isn't received by that time then it can assume that the process has failed. It implies that the system may mistake a delayed message for a process failure. In this case, we can't implement the algorithm we use for synchronous systems' scenario 3 and reach consensus. Because the number of assumed failed processes will be higher than f.

b) **[10 Points]** Devise a consensus algorithm for this new scenario. Explain the limitations your algorithm.

In a timed asynchronous system, we have a approximation of latency and an accuracy to which the system behaves. If n is the number of total processes and f is the number of failed processes then,

Number of rounds = f+1+(1-Accuracy of statistical analysis)*n

for example if analysis says that 99% messages get delivered before timeout then,
Number of rounds = f+1+(1-0.99)*n

But, if the statistical analysis is not very accurate and in the case where number of rounds exceeds n i.e. Number of rounds > n then this algorithm can not give results.

Other limitation is the assumption that the messages which were delayed more than the calculated approximate latency indicated a process failure. This may lead to a situation where a functioning process may have been assumed failed and it's value may have been disregarded during the consensus decision,

9. **[16 Points]** State the safety and liveness properties for the following applications.

a) Traffic light at a four-way intersection controlled by two processes. One process controls N-S direction lights and the other process controls the E-W direction lights.

Safety property - both green lights never gets turned on at the same time

Liveness property - In the case of a power outage, there is back-up power

b) A highly available airline reservation system that allows clients to book seats.

Safety property - never allow 2 users to book same seat

Liveness property - Every user eventually gets a reservation for a desired destination

c) An ATM that dispenses cash to the bank clients.

Safety property - Never give out wrong amount of cash

Liveness property - Frequent cash refills so it doesn't run out of cash and back-up power generator

d) An air traffic control system that manages plane landings and takeoffs.

Safety property - Always calculate the parameters correctly by using sensor data. Confirm it again and have human assistance from pilots

Liveness property - Redundant sensors to avoid failures and a co-pilot