Solutions - Mid Paper DS-21

Section 1 : Mutual Exclusion

Question 1)

(a) (2 points)Consider the Suzuki Kasami Algorithm. Let process Pi be the process currently holding the token. Process Pj has broadcasted a request for the token. Can process Pj enter CS before Pj's request reaches Pi? Argue.

yes, it is possible if Pi sends it token to some other process k and it releases to Pj before the original message reaches to Pi.

(b) Consider the Raymonds Algorithm for ME. Answer the following questions.

• (1 point) If the process holding the token is interested in entering CS again while holding the token; then, can he quietly just re-enter the CS? Argue.

I) yes, it can, but it would not be fair

• (1 point) Can the process who has received the token and completed critical section(current root)at time t; find his own id on top of his queue before the token travels again to some other process ? Argue.

Ii) yes, if system allows to make multiple request else no

• (1 point) Can the process who is executing the critical section (current root) have his own id also in his queue

Iii) No, as it removed during execution. However, if the system allows for multiple requests of CS then it is possible.

*If assumptions for 2nd and 3rd part are consistent then only you get marks. If assumption is explicitly not mentioned, then some marks are deducted (remember in the algo while requesting critical section it is nowhere mentioned that it cannot place multiple requests in the queue. Only if it receives a request, it cannot send to the parent again.)

(c) (2 points) Consider the Maekawas Algorithm. Prof Z changes to conditions in the Maekawas algorithm slightly as can be seen below

• for all i, j: Ri ∩Rj6 = ∅

• for all i: i ∈ Ri

• for all i: kRik = N , where N is the total number of nodes in the system

• Any node i is contained in exactly N request sets.

For such a change what would be your feedback to Prof Z.

No benefit of quorum based algos to reduce space and time complexity with quorum size as N

(d) (3 points) You are tasked with designing a new distributed mutual exclusion protocol based on voting. You come up with the following solution:

• A node requests permission (votes) from other nodes before proceeding to execute its critical section.

• The node does not proceed unless it receives a majority of replies from other nodes. What are some problems with this solution?

Some of the points:

- No ME as multiple can execute
- If multiple are requesting, how are a single node supposed to vote ? (fcfs?)
- Single vote per node?
- Deadlock if none gets the vote
- starvation
- How does new node know when he gets the vote or the vote expires (basically starvation)

(e) (5 points) Prof A and Prof B suggested modifications to the Suzuki Kasami algorithm for a process P that has the token (just finished a critical section) and is interested to re-enter the critical section. Assume that when this happens the token queue is not empty.

Prof A's suggestion: Process P puts itself on token queue and then sends the token to the next process on the queue.

Prof B's suggestion: P executes its second CS and then sends token to the next process in the token queue

• Mention why you think each of Prof A and Prof B felt it was worth including their suggestion

• Comment on whether you feel both the modifications give the desired output. If further modifications are required for the idea to work then mention so.

ProfA more fair, ProfB optimises and practical but only if it limits to 2 executions (how to maintain that?)

Question2) (15 points) Consider the below Sarangai-Weiss Algorithm. This algorithm involves counter values. A counter if sent is either sent immediately on receiving a REQUEST or after the critical section based on certain conditions. A counter sent after critical section is called C Counter and the one sent immediately on seeing a request is called A Counter. Initial values of A Counter and C Counter = 0 for all sites.
Also note that If two processes have the same time stamp then the tie is broken by using the process id.
Requesting the critical section:

1.When a site S_i wants to enter the CS, it broadcasts a time stamped REQUEST message to all other sites.
2.When site S_k receives a REQUEST message from site S_i, it sends a A Counter with value 1 to site S_i if

- site S_k is neither requesting nor executing the CS,
- or the site S_k is requesting and timestamp(S_i) < timestamp(S_k)

Else
the decision of replying or not is deferred to after the critical section.

Executing the critical section:

Site S_i enters the CS when the summation of all Counter values received (of both A Counters and C Counters together) adds to exactly n-1 (where n is the total number of sites). Note: It need not re- ceive a Counter from every site. As soon as the sum of received counters sums to n-1; he can enter the CS.

Releasing the critical section:

When site S_i exits the CS, it does the below in sequence:
- If he has not received a C Counter from any site then he sets his C Counter = 1 else he sets his C Counter as the received C counter value incremented by 1.
- sends the C Counter to a single process which is the next process with lowest time stamp.
- Resets his C Counter = 0

Algorithm Idea: Consider the sites interested in entering the critical section ordered by increasing timestamp be P1, P2, P3, ... Pi. (If two processes have the same time stamp then the tie is broken by using the process id). According the algorithm the site P2 will be waiting only on the C Counter of the site P1 to enter CS as every other site would have given its permission using A Counter = 1. The site P2 after its CS; will let the site P3 know and give it the combined permission of itself and P1 with C counter = 2 and so on.

Questions:

a. (3 points) What is the advantage of such an algorithm?

Reduces number of messages required to enter the critical section compared to Lamport and Ricart Agarwala algorithm. 2(N-1) worst case. N best case. Works on non-FIFO.

b. (8 points) Does this algorithm satisfy Mutual Exclusion, fairness, no-deadlock and no-starvation? Explain.

Mutual exclusion - yes - only 1 process can enter cs at a time, fairness (in-order (by logical time)) - yes, no deadlock - no, no starvation (bounded wait) - no. (explain) Once the c_counter crosses n-1 no one can enter the critical section.

c. (4 points) What would be any VALUABLE feedback you have about this algorithm?

way to correct the algorithm - sum >= n-1 instead of sum == n-1 and no need for A_counter OR pointing out some other mistake in the algorithm

Question3) (15 points) Consider the below Maria Seene Modified Algorithm. This algorithm involves counter val- ues. A counter if sent is either sent immediately on receiving a REQUEST or after the critical section based on certain conditions. A counter sent after critical section is called C Counter and the one sent immediately on seeing a request is called A Counter.

Initial values of A Counter and C Counter = 0 for all sites.
Also note that If two processes have the same time stamp then the tie is broken by using the process id.

Requesting the critical section:

1.When a site S_i wants to enter the CS, it broadcasts a time stamped REQUEST message to all other sites.
2.When site S_k receives a REQUEST message from site S_i, it sends a A Counter with value 1 to site S_i if

- site S_k is neither requesting nor executing the CS,
- or the site S_k is requesting and timestamp(S_i) < timestamp(S_k) Else
the decision of replying or not is deferred to after the critical section.

Executing the critical section:

Site S_i enters the CS when the summation of all Counter values received (of both A Counters and C Counters together) adds to greater than or equal to n-1 (where n is the total number of sites). Note: It need not receive a Counter from every site. As soon as the sum of received counters sums greater than or equal to n-1; he can enter the CS.

Releasing the critical section:

When site S_i exits the CS, it does the below in sequence:
- If he has not received a C Counter from any site then he sets his C Counter = 1 else he sets his C Counter as the received C counter value incremented by 1.
- sends the C Counter to a single process which is the next process with lowest time stamp.
- Resets his C Counter = 0

Questions:

a. (2 points) What is the advantage of such an algorithm?

Reduces number of messages required to enter the critical section compared to Lamport and Ricart Agarwala algorithm. 2(N-1) worst case. N best case. Works on non-FIFO.

b. (5 points) Does this algorithm satisfy Mutual Exclusion, fairness, no-deadlock and no-starvation? Explain.

Mutual exclusion - yes - only 1 process can enter cs at a time, fairness (in-order (by logical time)) - yes, no deadlock - yes, no starvation (bounded wait) - yes. (explain)

c. (5 points) Any issues you see with the algorithm and any comments on how you would modify it if you could.

sum of c_counter can become very large - cap it at n-1 and no need for A_counter OR pointing out some other mistake in the algorithm

d. (3 points) Compare and comment on this algorithm compared to the ME algorithms you have studied

Any comparison with any other algo taught in class regarding messaging, time or space complexity.

Section 2: Clock

Question 1) Kshemakalyani's differential technique for vector clocks:

a) **(3 marks)** Explain the construction of the extra vectors used in this technique.

Process pi maintains the following two additional vectors:

LSi [1...n] ('Last Sent'): LSi[j] indicates the value of vti[i] when process pi last sent a message to process pj. **(1.5 marks)**

LUi [1...n] ('Last Update'): LUi[j] indicates the value of vti[i] when process pi last updated the entry vti[j]. **(1.5 marks)**

b) **(2 marks)** What is the message passing condition? Justify.

Clearly, LUi [i] = vti [i] always and LUi [j] needs to be updated only when the receipt of a message causes pi to update entry vti [j]. Also, LSi [j] needs to be updated only when pi sends a message to pj .

c) **(4 marks)** Explain how it helps in decreasing storage than the naive method used for reduced message passing.

Since the last communication from pi to pj , only those elements of vector clock vti [k] have changed for which LSi [j] < LUi [k] holds.

Hence, only these elements need to be sent in a message from pi to pj . When pi sends a message to pj , it sends only a set of tuples {(x, vti [x]) | LSi [j] < LUi [x]} as the vector timestamp to pj , instead of sending a vector of n entries in a message.

Thus, the entire vector of size n is not sent along with a message. Instead, only the elements in the vector clock that have changed since the last message send to that process are sent in the format {(p1, latest value), (p2, latest value), . . .}, where pi indicates that the pith component of the vector clock has changed.

d) **(1 mark)** What is the limitation of this method?

This technique requires that the communication channels follow FIFO discipline for message delivery.

Question 2) Sheldon was trying to find a manner to get the scalar time of an event from its vector time. This was not possible with raw vector clock implementation. So he figured out that only if he had an extra integer C along with the vector time (vt), he would get the scalar time as a function of the vector vt and C. The extra term C has its initial value set to 0 for all processes and has the following update rule for a message send event from process Pi to Pj:

Ci := Ci

Cj := Cj + Ci + 1

Then according to Sheldon for a process Pi, the transformation from vector to scalar time would be:

Scalar Time = $\Sigma^{n}_{j=1} vt_i[j]$ - C

Is Sheldon correct? Justify your stance. (Assume step size d = 1 and the logical times start with 1, See example image for better understanding)

Answer) Sheldon is incorrect. (2 marks)

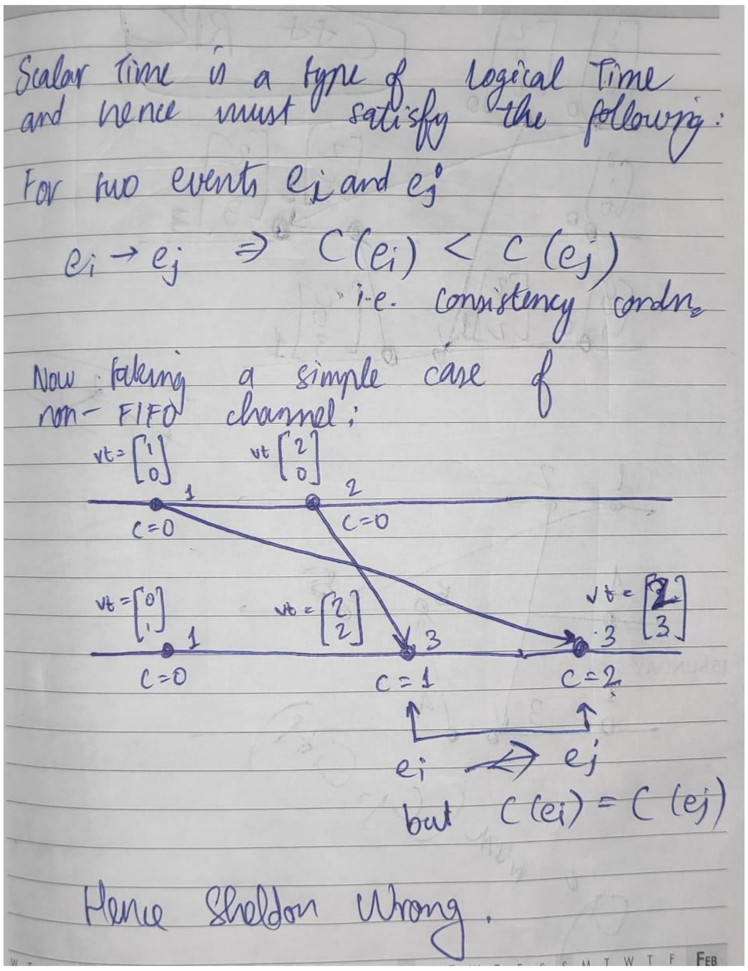
Proof by counter example. (8 marks)

Any other proof if valid gets full marks.

No steps shown results in penalty of 2 marks

The given algorithm only works in some cases for FIFO channels.

Even if answer wrong, mentions about consistency condition and knowledge about scalar and vector clocks fetches 1-2 marks.


Scalar time is a type of logical time and hence must satisfy the following:

For two events e_i and e_j

$e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$

i.e. consistency condn.

Now taking a simple case of non-FIFO channel:

$vt = \begin{bmatrix}1\\0\end{bmatrix}$
1
C=0
$vt \begin{bmatrix}2\\0\end{bmatrix}$
2
C=0
$vt = \begin{bmatrix}0\\1\end{bmatrix}$
1
C=0
$vt = \begin{bmatrix}2\\2\end{bmatrix}$
3
C=1
$vt = \begin{bmatrix}2\\3\end{bmatrix}$
3
C=2

$e_i \not\rightarrow e_j$

but $C(e_i) = C(e_j)$

Hence Sheldon Wrong.

Question 3) Numericals on logical clocks:

a) There are 3 processes having K number of events each where K = X mod 3 + 4. Now in this system 3*(K-1) total number of message passing events are involved (one message pass event constitutes a message sent event by a process and a message receive event by another process). Find the minimum and maximum value of scalar time that is possible and show the order of message passing with a diagram. Take d = 1 and initial scalar time for the first event of each process to be by default 1.

Answer) Minimum Scalar Time : K (1.5 marks)

Maximum Scalar Time : 3K (1.5 marks)

Min Diagram (1 mark) Max Diagram (1 mark)

Use of incorrect K or misunderstanding has penalty of 2 marks

Observed that a lot of students are considering event 1 of each process to have scalar time as strictly 1 while it was mentioned as default. Giving them the leeway and hence giving complete marks for max scalar time being 3K – 2.

b) For vector clock with step size as d (which is given by d = X mod 2 + 2) , what is the relation between the vector times and the total number of events causally preceding it. Assume that the number of processes are n and the logical time begins with 1 for each process. For a given vector time vt_i you can consider the number of zeros in it to be k_i .

Answer) Complete answer:

$((d-1)(n-k_i) + \Sigma^n_{j=1} vt_i[j])/d - 1$ (4 marks)

1 mark for mentioning equation: Number of causally preceding events = $\Sigma^n_{j=1} vt_i[j] - 1$

Close but incorrect answers to this equation gets 2 marks.

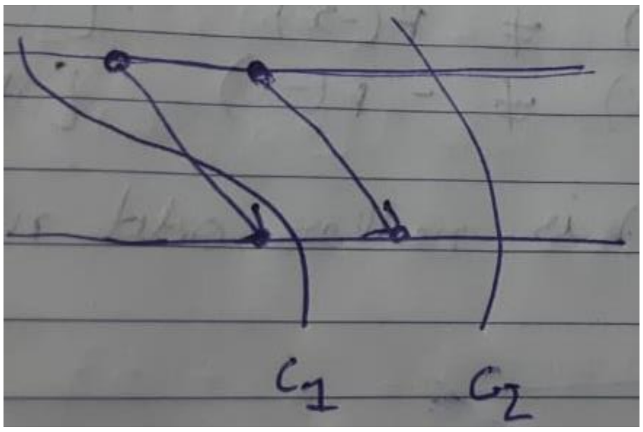
Section 3 : Global Snapshots

Question 1) Consider two cuts whose events are denoted by C1=C1(1),C1(2),.... ,C1(n) and C2= C2(1),C2(2),.... ,C2(n) respectively. (Note: In the above, Ci(j) for integers i and j refer to the local time that cut Ci intercepts the timeline of processor Pj). Define a third cut,C3= C3(1),C3(2),....,C3(n), which is the minimum of C1 and C2; that is, for every k, C3(k) is the earlier of C1(k) and C2(k). Then

Given two cuts : C1, C2 , we define C3 = min(C1, C2)

(a) will cut C3 be a consistent cut ? Prove or disprove.

Proof by counter example



C3 = min(C1, C2) = C1 = inconsistent, thus C3 is not always consistent

Could also explain for different cases

MARKING : (Max Marks = 3)

mentioned inconsistency (1 mark); correct reasoning (2 marks); partially correct reasoning(1 mark)

(b) will cut C3 be consistent if C1 and C2 are consistent

Yes, C_(k+1) must be consistent. Let us assume C_(k+1) is inconsistent. If C_(k+1) is inconsistent then for one message 'm', receive(m) is captured by C_(k+1) and send(m) is not captured by C_(k+1). Also as C_(k+1) = min (C1, C2, C3, …., Ck), there must be a cut Ci which captures receive(m) but does not capture send(m) [because we take minimum of all], this makes Ci inconsistent => 'contradiction!' as mentioned all cuts Ci are consistent.

MARKING: (Max Marks = 4)
Mentioned about consistency of C3 (1 mark); correct reasoning (3 marks); partially correct reasoning (1-2 marks)

(c) What can you say if we have k cuts C1 through Ck?

Using part a and b

MARKING : (Max Marks = 3)
Mentioned about consistency/inconsistency along with the specific case (1 mark); correct reasoning (2 marks); partially correct reasoning (1 marks)

Question 2) Snapshot Algorithms:

a) For the following systems state all the global snapshot algorithms (out of Chandy Lamport, Lai Yang and Acharya Badrinath) that could be applied with proper justification (You can safely assume all the algorithms to work in case of a lack of information)

I) any would work with causal ordering
Ii) same as (i)
Iii) non fifo so Lai Yang only
Iv) 4th is non Causal Ordering but it could be fifo or non-fifo so Chandy Lamport or Lai Yang

b) Consider a distributed system where every node has its physical clock and all physical clocks are perfectly synchronized. Give an algorithm to record global state assuming the communication network is reliable. (Note that your algorithm should be simpler than the Chandy–Lamport algorithm.)

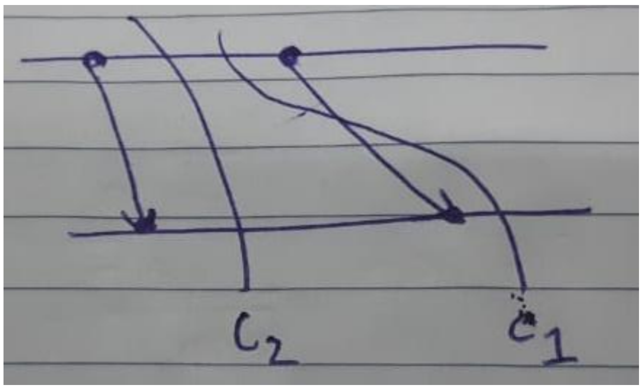
Any Simple Algo like where you decide on a time(in future) and everyone records snapshot at that time would work.

Question 3) Consider two cuts whose events are denoted by C1=C1(1),C1(2),.... ,C1(n) and C2= C2(1),C2(2),.... ,C2(n) respectively. (Note: In the above, Ci(j) for integers i and j refer to the local time that cut Ci intercepts the timeline of processor Pj). Define a third cut,C3= C3(1),C3(2),....,C3(n), which is the maximum of C1 and C2; that is, for every k, C3(k) is the later of C1(k) and C2(k). Then

Given two cuts : C1, C2 , we define C3 = max(C1, C2)

(a) will cut C3 be a consistent cut ? Prove or disprove.

Proof by counter example



C3 = max(C1, C2) = C1 = inconsistent

Could also explain for different cases

MARKING : (Max Marks = 3)

mentioned inconsistency (1 mark); correct reasoning (2 marks); partially correct reasoning(1 mark)

(b) will cut C3 be a consistent if C1 and C2 are consistent

Yes, any message received in C_(k+1) must be received in some Ci (0<i<k+1). Also atleast one cut Cj(0<j<k+1 ; including i as Ci is consistent) must capture the corresponding send and as C_(k+1) is max of all cuts, it must also record the corresponding send. Hence C_(k+1) is always consistent.

MARKING: (Max Marks = 4)
Mentioned about consistency of C3 (1 mark); correct reasoning (3 marks); partially correct reasoning (1-2 marks)

(c) What can you say if we have k cuts C1 through Ck?

Using part a and b

MARKING : (Max Marks = 3)
Mentioned about consistency/inconsistency along with the specific case (1 mark); correct reasoning (2 marks); partially correct reasoning (1 marks)

Section 4: Presentations

Marking -> 0 if explanation not given in True/False

1) (8 points) Fill in the blanks (3 points):

a) In Birman Schiper Stephenson Algorithm, all messages are **Broadcast** messages.

b) In Spezialetti-Kearns algorithm, each process keeps track of the initiator using the **Master** variable.

c) Alagar-Venkatesan algorithm requires **3**-time units and **3** * n messages. (n is #processes).

State True or False with explanation (5 points):

(a) Spezialetti-Kearns algorithm will work for Causal Channels.

Ans) True, an algorithm for FIFO channel would work for causal channel as well

(b) In Spezialetti-Kearns algorithm, the number of regions might be less than the number of concurrent initiators.

Ans) False, it should be equal to number of concurrent initiators.

(c) Message complexity of Singhal's dynamic information structure algorithm can be

5/4 * (n - 1) where n is total number of sites.

Ans) True, it can be anything between (n-1) and 3 * (n – 1) / 2.

(d) In Singhal's Dynamic Information-Structure Algorithm, the REQUEST and REPLY code sections cannot run concurrently as they both try to modify the logical clock.

Ans) False, R and I sets are modified.

(e) Singhal's Dynamic Information-Structure Algorithm will work for non-FIFO channels.

Ans) False, it is designed for FIFO channel

2) (8 points) Fill in the blanks (3 points):

(a) In Spezialetti-Kearns algorithm, **id-border-set** variable of a process contains the identifiers of the neighbouring regions.

(b) In Mattern's algorithm for Non-FIFO channel, **Termination Detection scheme/Termination Detection** is used to detect that no white messages (sent before snapshot) are in transit.

(c) The synchronisation delay in Singhal's dynamic information structure algorithm is **1** * T.

State True or False with explanation (5 points):

(a) Mattern's Algorithm will work for FIFO channels.

Ans) True, an algorithm for non-FIFO channel would work for FIFO channel as well

(b) Lai-Yang algorithm uses more memory as compared to Mattern's Algorithm.

Ans) True, in Mattern's algorithm, a process is not required to store message histories to evaluate the channel states.

(c) A quorum in Agarwal El-Abbadi Algorithm consists of all processes in the path from root to a leaf excluding the root.

Ans) False, it includes root as well

(d) In Agarwal El-Abbadi algorithm, if a non-leaf node p is unresponsive, permission can be asked from all processes on two paths instead: from each child of p to some leaf.

Ans) True (Reason not required for this part)

(e) Agarwal El-Abbadi algorithm can guarantee formation of a quorum even if number of site failures is between 2 * log(n) and 3 * log(n) where n is total number of sites.

Ans) False, it should be at max log(n).