The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 2: Distributed Clocks

Notes

Please ensure that you make every effort to have your camera on and actively participate in the tutorial discussion. Exercise questions marked * are additional questions that won't be the focus of the tutorial but may be useful in your revision for quizzes and the final exam.

The tute sheet for the **following week** and solutions for the **current week** will be provided weekly on Friday afternoon.

Modern computers usually have two different types of clocks:

1. **Time of Day Clocks:** *clock_gettime(CLOCK_REALTIME)*
2. **Monotonic Clocks:** *clock_gettime(CLOCK_MONOTONIC)*

Exercises

1. Consider the following: A clock is reading 10:27:54.0 (hr:min:sec) when it is discovered to be 4 seconds fast.

(a) What condition would be broken if the clock were immediately set back to 10:27:50.0?

(b) Show how the clock should be adjusted as to be correct after at 10:28:02.0.

2. A scheme for implementing at-most-once reliable message delivery uses synchronized clocks to reject duplicate messages. Processes place their local clock value (a 'timestamp') in the messages they send. Each receiver keeps a table giving, for each sending process, the largest message timestamp it has seen. Assume that clocks are synchronized to within 101 ms, and that messages can arrive at most 50 ms after transmission

(a) When may a process ignore a message bearing a timestamp T, if it has recorded the last message received from that process as having timestamp T'?

(b) When may a receiver remove a timestamp 175,000ms from its table?

(c) Should the clocks be internally or externally synchronized?

3. A client attempts to synchronize with a time server. It records the round trip time T_{round} and timestamp t returned by the server. In some cases it knows T_{min}, the minimum possible delay between sending and receiving a message. What time should the client set it's clock to? What is the accuracy in respect to the time server?

(a) $T_{round} = 4$s, $t =$10:55:22

(b) $T_{round} = 3$s, $t =$10:55:22, $T_{min} = 1$s

4. Assuming the minimum delay is 1 second, using Christian's algorithm, what must be the case in order to achieve an accuracy of ± 1s with respect to the time server?

5. An NTP server B receives server A's message at 16:34:23.480 bearing a timestamp 16:34:13.430 and replies to it. A receives the message at 16:34:15.725, bearing B's timestamp 16:34:25.7. Estimate the offset between B and A and the accuracy of the estimate.

6. Show that for a system with an external synchronization bound of D, the internal synchronization is bounded by $2 \cdot D$*

7. Discuss how it is possible to compensate for clock drift between synchronization points by observing the drift rate over time. Discuss any limitations to your method.*

8. Discuss the factors to be taken into account when deciding to which NTP server a client should synchronize its clock.*

9. What reconfigurations would you expect to occur in the NTP synchronization subnet?*

Questions based off Colouris, Dollimore and Kindberg

THE UNIVERSITY OF MELBOURNE
SCHOOL OF COMPUTING AND INFORMATION SYSTEMS
COMP90020 DISTRIBUTED ALGORITHMS

Tutorial Week 3: Logical Clocks

Notes

Clock Condition. For all events a, b: if $a \to b$ then $C(a) < C(B)$. This is satisfied if the following two conditions hold:

- $C1$. If a and b are events in process p_i and a comes before b, then $C_i(a) < C_i(b)$
- $C2$. If a is the sending of a message by process P_i and b is the receipt of that message by process P_j, then $C_i(a) < C_j(b)$

Exercises

10. This question is from last weeks tutorial, please attempt if you haven't already. An NTP server B receives server A's message at 16:34:23.480 bearing a timestamp 16:34:13.430 and replies to it. A receives the message at 16:34:15.725, bearing B's timestamp 16:34:25.7. Estimate the offset between B and A and the accuracy of the estimate.

11. Consider the following sequence of events at processes p_0, p_1, p_2 and p_3. Here s_i and r_i are corresponding send and receive events for all i, while a and b are internal events.

$$\begin{array}{llllll}
p_0: & s_1 & s_2 & r_5 & & \\
p_1: & r_2 & s_5 & & & \\
p_2: & r_1 & a & s_4 & r_3 & r_6 \\
p_3: & s_3 & r_4 & b & s_6 &
\end{array}$$

Use Lamport's logical clock to assign clock values to these events (A diagram may help).

12. By considering a chain of zero or more messages connecting events e and e' and using induction, show that $e \to e' \Rightarrow L(e) < L(e')$

13. Using the same sequence of events from exercise *10*, draw a send/receive diagram to give Vector timestamps to each of the events.

14. Show that:

(a) $V_j[i] \leq V_i[i]$ for all i, j

(b) $e \to e' \Rightarrow V[e] < V[e']$

(c) $V(e) < V(e') \Rightarrow e \to e'$

THE UNIVERSITY OF MELBOURNE
SCHOOL OF COMPUTING AND INFORMATION SYSTEMS
COMP90020 DISTRIBUTED ALGORITHMS

Tutorial Week 4: Global States

Notes

An event is called *presnapshot* if it occurs at a process before the local snapshot at that process is taken; otherwise it is called *postsnapshot*. A snapshot is consistent if

- (1) no postsnapshop event is causally before a presnapshot event
- (2) a basic message is included in a channel state if and only if the corresponding send event is presnapshot while the corresponding receive event is postsnapshot.

Exercises

15. Consider the following figure:

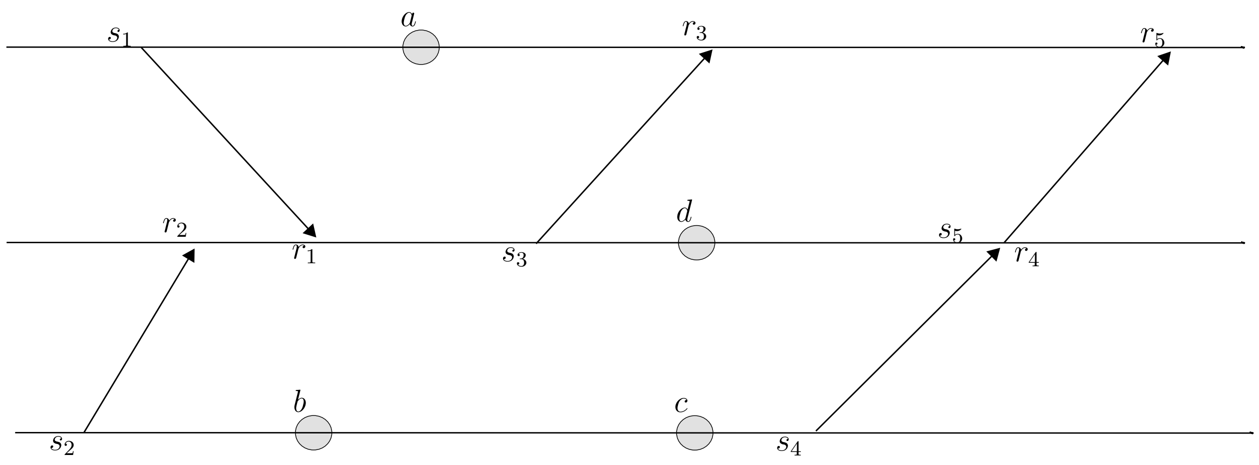


Figure 1: Three process sequence diagram

(a) Provide the vector timestamps at each send, receive and concurrent event.

(b) Provide an example of a consistent and inconsistent cut.

16. Consider the following: Two processes P and Q are connected in a ring using two channels, and they constantly rotate a message m. At any one time, there is only one copy of m in the system. Each process's state consists of the number of times it has received m, and P sends m first. At a certain point, P has the message and its state is 102. Immediately after sending m, P initiates the chandy-lamport snapshot algorithm. Explain the operation of the algorithm in this case, giving the possible global state(s) reported by it.

17. Consider the following figure

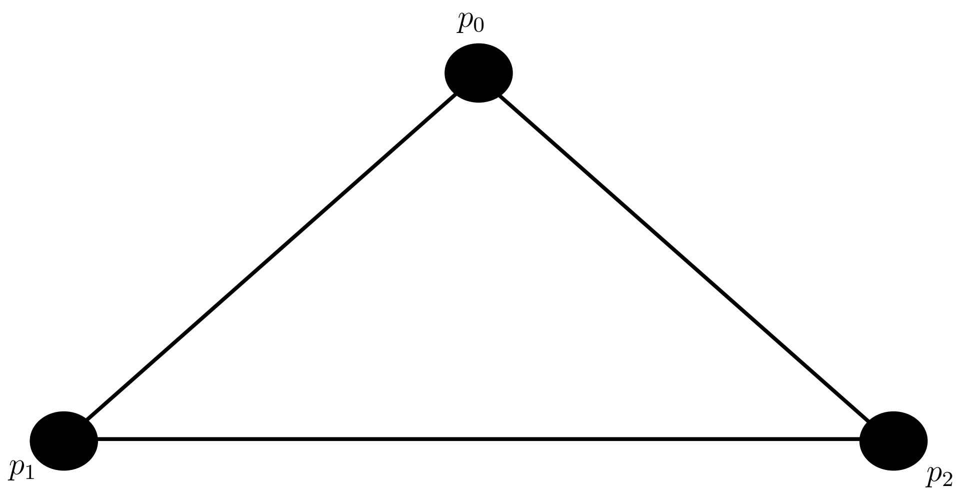


Figure 2: Channel complete communication with three processors

In the above system, each process calculates it's next message it will send, using the first message that exists in it's buffer as follows.

$$next(msg_i) = \begin{cases} \frac{msg_i}{2} & msg_i \bmod 2 = 0 \\ 3 \cdot msg_i + 1, & msg_i \bmod 2 = 1 \end{cases}$$

Likewise, each process m decides which corresponding neigbour k to send this message to as follows:

$$target(msg_i, m) = \begin{cases} (m+1) \bmod 3 & msg_i \bmod 2 = 0 \\ (m-1) \bmod 3, & msg_i \bmod 2 = 1 \end{cases}$$

Assume each node holds no local state, only processing it's buffer in a FIFO manner. Initially, each node has it's buffer contain a single value:

$$p_0 = \{11\}, p_1 = \{10\}, p_2 = \{9\}$$

(a) After some sequence of messages, each nodes buffer is as follows: $p_0 : \{\}, p_1 : \{\} p_2 : \{\{34\}\{5, 14\}\{\}\}$. Is this a possible (globally consistent) state?

(b) Assume in the above scenario, process p_0 initiates the chandy-lamport snapshot algorithm. Explain the operation of the algorithm in this case, giving the possible global state(s) reported by it.

The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 5: Mutual Exclusion and Failure Detection

Notes

In this tutorial we will look at mutual exclusion and briefly at failure detectors. Both of these topics have much more to offer. In particular, we will only examine mutual exclusion using message passing and won't look at algorithms that use shared memory.

If you're interested to learn more broadly on these topics, a great place to start is Lamport's bakery algorithm and further continue onto queue, CLH and MCS locks.

Exercises

18. Is it possible to implement either a reliable or an unreliable (process) failure detector using an unreliable communication channel?

19. Say for both mutual exclusion and startvation-freeness whether it is a safety or liveness property.

20. In the central server algorithm for mutual exclusion, describe a situation in which two requests are not processed in happened-before order.

21. Give an example execution of the ring-based algorithm to show that processes are not necessarily granted entry to the critical section in happened-before order

22. Consider running Maekawa's algorithm on a system of 13 processes. Figure out the composition of the 13 subsets S_0 - S_{12}, so that (1) each subset includes four processes, (2) there are exactly four subsets, and (3) process $i \in S_i$.

23. The L-exclusion problem is a generalized version of the mutual exclusion problem in which up to L processes $L \geq 1$ are allowed to be in their critical sections simultaneously. Precisely, if fewer than L processes are in the CS at any time and one more process wants to enter its critical section, then it must be allowed to do so. Modify Ricart–Agrawala's algorithm to solve the L-exclusion problem.

24. In a network of processes, the *local mutual exclusion* problem guarantees that no two neighbors execute a critical action at the same time. Extend Ricart–Agrawala's to solve the local mutual exclusion problem.

The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 6: Leader Election

Notes

Exercises

25. In the Bully algorithm, a recovering process starts an election and will become the new coordinator if it has a higher identifier than the current incumbent. Is this a necessary feature of the algorithm?

26. Suggest how to adapt the Bully algorithm to deal with temporary network partitions (slow communication) and slow processes.

27. Assume that processes do not have a unique identifier (as previously assumed) and are instead anonymous. Is there a deterministic election algorithm for leader election in a ring of $n > 1$ processes? Provide a counter-example or provide such an algorithm..

28. The problem of leader election has some similarities with the mutual exclusion problem. Last week we discussed Maekawa's distributed mutual exclusion algorithm with $O(\sqrt{n})$ message complexity. Can we use similar ideas to design a leader election algorithm with sub-linear message complexity?

29. In a hypercube of n nodes, suggest an algorithm for leader election with a message complexity of $O(n \cdot \log n)$.

The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 7: Multicast

Notes

Exercises

30. How, if at all, should the definitions of integrity, agreement and validity for reliable multicast change for the case of open groups?

31. Consider the following algorithm from the textbook:

```
On initialization
    Received := {};

For process p to R-multicast message m to group g
    B-multicast(g, m);          // $p \in g$ is included as a destination

On B-deliver(m) at process q with g = group(m)
    if (m $\notin$ Received)
    then
                Received := Received $\cup$ {m} ;
                if (q $\neq$ p) then B-multicast(g, m); end if
                R-deliver m;
    end if
```

Consider reversing the order of the lines:

if ($q \neq p$) then B-multicast(g,m) end if
R-Deliver m

(a) Why would this change make the algorithm no longer satisfy uniform agreement?

(b) Does the reliable multicast algorithm based on IP multicast satisfy uniform agreement?

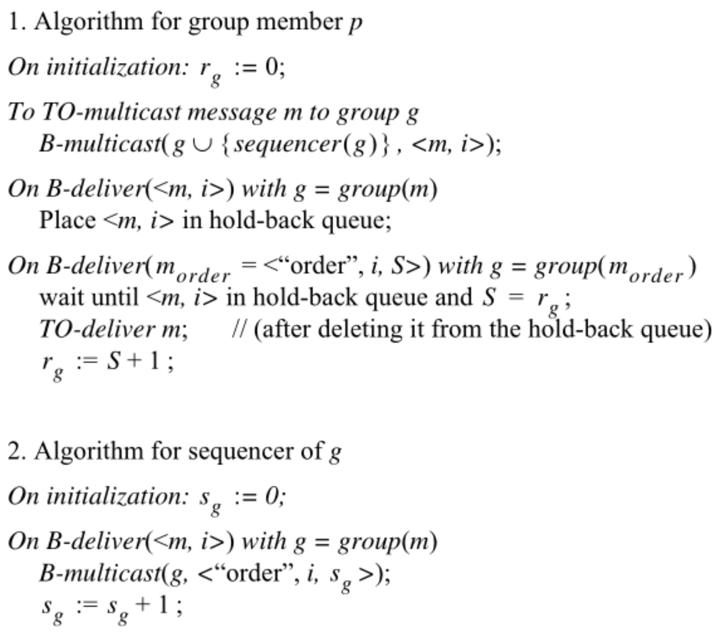
32. Show that the FIFO-ordered multicast algorithm does not work for overlapping groups, by considering two messages sent from the same source to two overlapping groups, and considering a process in the intersection of those groups. Adapt the protocol to work for this case. Hint: processes should include with their messages the latest sequence numbers of messages sent to all groups.

33. In a group, there are eight members, of which at most one can crash at any time. You have no idea about which process can crash. The topology is a completely connected network.

(a) In the worst case, what is the smallest number of messages required to guarantee reliable atomic multicast from a given process to the entire group?

(b) How does this change if up to four members can crash at any time?

34. Consider the following algorithm from the textbook:



```
1. Algorithm for group member p
On initialization: $r_g$ := 0;
To TO-multicast message m to group g
    B-multicast( $g \cup \{sequencer(g)\}$ , <m, i>);
On B-deliver(<m, i>) with g = group(m)
    Place <m, i> in hold-back queue;
On B-deliver( $m_{order}$ = <"order", i, S>) with g = group( $m_{order}$ )
    wait until <m, i> in hold-back queue and $S = r_g$ ;
    TO-deliver m;      // (after deleting it from the hold-back queue)
    $r_g$ := S + 1 ;

2. Algorithm for sequencer of g
On initialization: $s_g$ := 0;
On B-deliver(<m, i>) with g = group(m)
    B-multicast(g, <"order", i, $s_g$ >);
    $s_g$ := $s_g$ + 1 ;
```

(a) Show that, if the basic multicast that we use in the algorithm is also FIFO-ordered, then the resultant totally-ordered multicast is also causally ordered.

(b) Is it the case that any multicast that is both FIFO-ordered and totally ordered is thereby causally ordered?

35. Consider a multiparty game of quiz involving five teams.

Each team poses a question and a member of another team has to answer it. The team that answers the question first scores a point. If no team can answer a question within 30s, then no one scores any point. The clocks are synchronized, so who answered first is decided by the time when the reply was posted. The teams can pose questions at any time and in no particular order.

What kind of multicast is appropriate here?

36. [EXTENSION] Consider an election in a state. The citizens cast their votes at the individual polling centers. At the end of the day when the poll closes, counting begins. Each count recorded at a center is multicast to all the other centers, so that all of them exactly know the latest count at any time when the counting is in progress. The interconnection topology of the network connecting the polling centers is a completely connected graph.

(a) Assume that at any time failures can bring down some of the communication lines without creating a partition. What kind of multicast will guarantee that all centers are able to record every vote?

(b) Assume that communication failure partitioned the network for an hour. Apparently the counting must stop. What would you recommend so that the progress of counting is not affected even if the network temporarily partitions?

The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 8: Distributed Consensus

Notes

Note all of the following assume the system is synchronous (why?). Coordinated attack assumes possible channel faults. Byzantine Agreement and Byzantine Broadcast assume possible byzantine (arbitrary) faults.

Coordinated Attack

We have $n \geq 2$ nodes. The messages received in round $i+1$ is always a subset (possibly empty, communication unreliable) of the set sent in round i. Each node starts with an input $v_i \in \{0,1\}$ and must decide on $d_i \in \{0,1\}$ in a bounded number of rounds.

- **Agreement:** $d_i = d_j$ for all nodes.
- **Integrity:** If $v_i = v_j$ for all pairs of nodes and no messages are lost, all processes have the same output $d_i = d_j$.
- **Termination:** All processes terminate in a bounded number of rounds.

Byzantine Broadcast (BB)

Some node ($p_j \in p_1, ..., p_n$) broadcasts v_j to every node ($\forall p_i \in \{p_1, ..., p_n\}$), and each correct node must agree on the message received. If p_j is correct then all correct nodes must agree $d_i = v_j$.

- **Agreement:** $d_i = d_j$ for each correct node.
- **Integrity:** If p_j is a correct, then all correct nodes must agree on ouput $d_i = v_j$.
- **Termination:** eventually $d_i \neq \perp$ for each correct node where initially $d_i = \perp$.

Commonly described in terms of Generals and lieutenants.

Byzantine Agreement (BA)

Each node ($\forall p_i \in p_1, ..., p_n$) begins holding an input v_i and they must come to an an agreement on their output.

- **Agreement:** $d_i = d_j$ for all correct nodes.
- **Integrity:** If $v_i = v$ for all correct nodes, then $d_i = v$ for all correct nodes.
- **Termination:** eventually $d_i \neq \perp$ for each correct node where initially $d_i = \perp$.

Exercises

36. Two loyal generals are planning to coordinate their actions for conquering a strategic town. To conquer the town, they need to both agreee on: attack or retreat; otherwise, if only one of them attacks and the other does not, then the generals are likely to be defeated. To plan the attack, they send messages back and forth via trusted messengers. The communication is synchronous. However, the messengers can be killed or captured so the communication is unreliable.

In this example is it possible to satisfy *agreement*, *integrity* and *termination* as stated in **Coordinated Attack**? Provide justification.

37. Construct an execution of the EIG algorithm to solve byzantine agreement (BA) where there are 3 processes, 1 process exhibits byzantine faults and integrity is not satisfied (draw two EIG trees and show they have a different λ).

38. When considering Byzantine Agreement (BA) and Byzantine Broadcast (BB):

(a) Is it possible to construct a solution to BB using a solution to BA? Briefly explain.

(b) Is it possible to construct a solution to BA using a solution to BB? Briefly explain (assume $f < \frac{n}{2}$).

39. Seven members of a family interviewed a candidate for the open position of a cook. If the communication is purely asynchronous and message-based, and decisions are based on majority votes, then describe a scenario to show how the family can remain undecided, when one member disappears after the interview.

40. Using oral messages in the context of generals reaching consensus on an attack where each p_i initially has some $v_i \in \{0, 1\}$, a solution to byzantine agreement (BA) can reach consensus when less than one-third of the generals are traitors (byzantine). However, it does not suggest how to identify the traitors. Examine if the traitors can be identified without any ambiguity if the number of traitors is known.

The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 9: Transactions and Concurrency Control

Notes

Strict executions of transactions: Generally, it is required that transactions delay both their read and write operations so as to avoid both dirty reads and premature writes. The executions of transactions are called strict if the service delays both read and write operations on an object until all transactions that previously wrote that object have either committed or aborted. The strict execution of transactions enforces the desired property of isolation. [Coulouris pp.690]

Exercises

42. A server manages the objects $a_1, a_2, ..., a_n$. The server provides two operations for its clients:

- $read(i)$: returns the v of a_i
- $write(i, v)$: assigns v to a_i

The transactions T and U are defined as follows:

- $T : x = read(j); y = read(i); write(j, 44); write(i, 33);$
- $U : x = read(k); write(i, 55); y = read(j); write(k, 66).$

(a) Give serially equivalent interleavings of T and U that are strict

(b) Give serially equivalent interleavings of T and U that are not strict but **could not produce** cascading aborts

(c) Give serially equivalent interleavings of T and U that **could produce** cascading aborts

43. The transfer transactions of T and U are defined as:

- $T : a.withdraw(4); b.deposit(4);$
- $U : c.withdraw(3); b.deposit(3);$

Suppose that they are structured as a pair of nested transactions

- $T_1 : a.withdraw(4); T_2 : b.deposit(4);$
- $U_1 : c.withdraw(3); U_2 : b.deposit(3);$

(a) Compare the number of serially equivalent interleavings of T_1, T_2, U_1, U_2 with the number of serially equivalent interleavings of T and U.

(b) Explain why the use of these nested transactions generally permits a larger number of serially equivalent interleavings than non-nested ones.

44. Consider the recovery aspects of the nested transactions defined in the exercise above, assume that a withdraw operation will abort if the account will be overdrawn and that in this case the parent will also abort.

(a) Describe serially equivalent interleavings of T_1, T_2, U_1, U_2 that are strict

(b) Describe serially equivalent interleavings of T_1, T_2, U_1, U_2 that are not strict

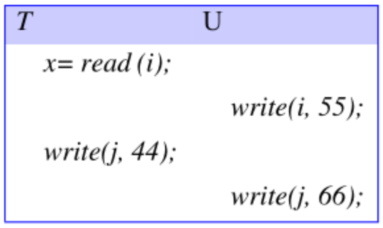
(c) To what extent does the criterion of strictness reduce the potential concurrency gain of nested transactions?

45. The transactions T and U are defined as follows:

- $T : x = read(i); write(j, 44);$
- $U : write(i, 55); write(j, 66);$

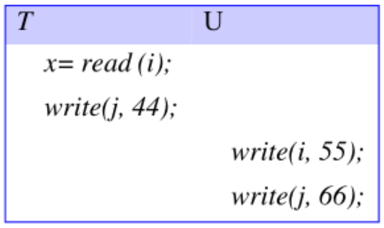
Initial values $a_i = 10, a_j = 20$. Which of the following interleavings are serially equivalent, and which could occur with two-phase locking?

(a)



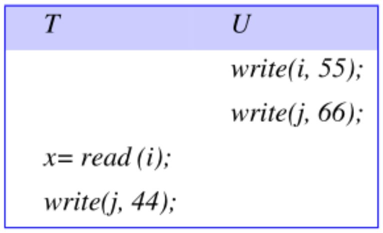
T	U
x= read (i);	
	write(i, 55);
write(j, 44);	
	write(j, 66);

(b)



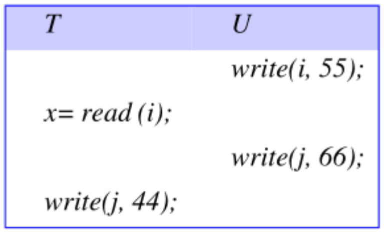
T	U
x= read (i);	
write(j, 44);	
	write(i, 55);
	write(j, 66);

(c)



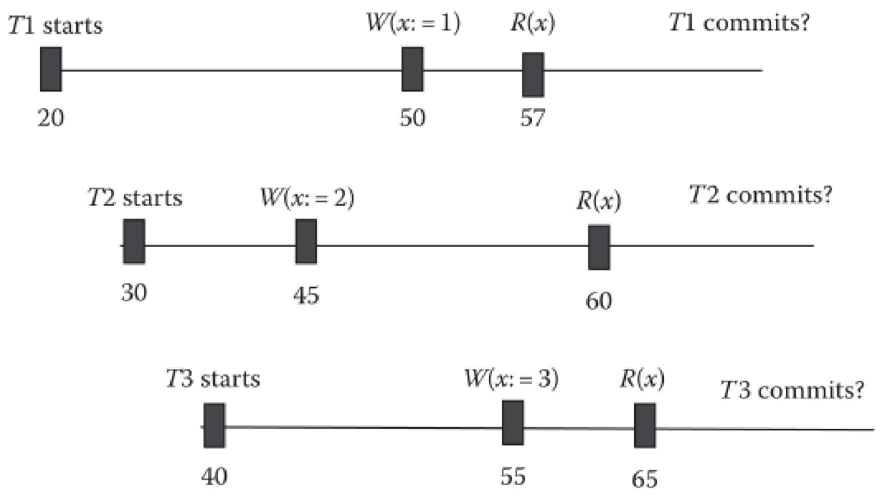
T	*U*
	write(i, 55);
	write(j, 66);
x= read (i);	
write(j, 44);	

(d)



T	*U*
	write(i, 55);
x= read (i);	
	write(j, 66);
write(j, 44);	

46. Consider three concurrent transactions below



Consider concurrency control by timestamp ordering. Which of these three concurrent transactions will commit?

THE UNIVERSITY OF MELBOURNE
SCHOOL OF COMPUTING AND INFORMATION SYSTEMS
COMP90020 DISTRIBUTED ALGORITHMS

Tutorial Week 11: Transactions and Concurrency Control Continued.

Notes

Two Phase Locking will be referred to as **2PL**.

Exercises

46. For each of the following input schedules, will deadlocking occur if 2PL is used? Provide the locking schedule. Note that $R(x, T_1), W(x, 5, T_2)$ represents a read in transaction 1 on object x and writing 5 into the object x in transaction 2 respectively.

(a) $R(x, T_1), W(y, 9, T_2), W(x, 4, T_2), W(y, 5, T_1)$

(b) $W(y, 9, T_2), R(x, T_1), W(x, 8, T_1), R(y, T_2)$

(c) $R(x, T_1), W(y, 0, T_2), R(x, T_2), W(x, 8, T_1)$

47. Consider a relaxation of two-phase locks in which read-only transactions can release read locks early.

(a) Would a read-only transaction have consistent retrievals?

(b) Would the objects become inconsistent?

48. Under 2PL protocol it is possible for a transaction to ”starve” in the following sense: A transaction gets involved in a deadlock, is chosen as the victim and aborted. After a restart, it again gets involved in a deadlock, is chosen as the victim and aborted, and so on. Provide a concrete example for such a situation, and describe how 2PL could be extended in order to avoid starvation of transactions.

49. Can the scenario described in 48 occur when timestamp ordering is used? Provide a concrete example or show why not.

50. Conservative or static 2PL (C2PL) is a variant of 2PL under which each transaction sets all locks that it needs in the beginning, before executing its first read or write steps. This is also known as preclaiming all necessary locks up front.

(a) Are deadlocks possible under this variant?

(b) If the read and write sets (all read and write operations) are not known by the scheduler ahead of time, is C2PL possible? Explain.

The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 12: Distributed Transactions.

Notes

Exercises

51. A three-phase commit protocol has the following parts:

- *Phase 1:* Is the same as for two-phase commit.
- *Phase 2:* The coordinator collects the votes and makes a decision. If it is No, it aborts and informs participants that voted Yes; if the decision is Yes, it sends a preCommitrequest to all the participants. Participants that voted Yes wait for a preCommitor doAbortrequest. They acknowledge preCommitrequests and carry out doAbortrequests.
- *Phase 3:* The coordinator collects the acknowledgements. When all are received, it commits and sends doCommitrequests to the participants. Participants wait for a doCommitrequest. When it arrives, they commit.

Explain how this protocol avoids delay to participants during their 'uncertain' period due to the failure of the coordinator or other participants. Assume that communication does not fail.

52. Give an example of the interleavings of two trnasactions that is serially equivalent at each server but is not serially equivalent globally.

53. Extend the definition of two-phase locking to apply to distributed transactions. Explain how this is ensured by distributed transactions using strict two-phase locking locally.

54. In the edge chasing algorithm, every transaction involved in a deadlock cycle can cause deadlock detection to be initiated. The effect of several transactions in a cycle initiating deadlock detection is that detection may happen at several different servers in the cycle, with the result that more than one transaction in the cycle is aborted. Explain a possible solution to this problem.

55. Consider a binary, perfectly balanced tree of processes of height n where all leaf nodes have the same distance from the root; so there is a total number of $m = 2^n - 1$ nodes in the tree. Assume that the root is the coordinator of the commit protocol. Determine the number of messages and forced log writes for the presumed nothing (2PC) protocol in the following situations, assuming that there is a heirarchic communication organization:

(a) all processes have performed updates and the transaction commits.

(b) all processes have performed updates and the transaction aborts (root decides to abort after receiving 2 YES).