HW4 - James Hall

Wednesday, October 31, 2012
10:14 PM

1. the Comparison Property and the "happened before" relation
   Given $\boldsymbol{a \rightarrow b \Leftrightarrow C\langle a\rangle < C\langle b\rangle}$, what can be inferred about the Logical Clock values if $\boldsymbol{a \parallel b}$?
   Nothing. Two events are concurrent if $a \nrightarrow b \wedge b \nrightarrow a$. Therefore, the relationship between $C\langle a\rangle$ and $C\langle b\rangle$ is unimportant.
2. the BSS algorithm and safety
   What could the impact be on a distributed filesystem that uses the BSS algorithm for message delivery if safety were violated?
   If safety were violated, a message from process P to Q could be overtaken by a message from process R to Q, which was sent later. If that happens, several problems could occur in a distributed filesystem. Such as deleting a file that doesn't exist, modifying a file that has already been deleted or doesn't exist, applying modification in the wrong order, or copying an incomplete file.
3. the BSS algorithm and liveness
   Assume a system has been up and running for an arbitrary amount of time. What would the impact on the system be if liveness were violated?
   The system may not continue after a certain point. Since some message may never be delivered, execution may stop at some arbitrary point in time, and nothing would continue after that.
4. Chandy's global-state-detection algorithm and inconsistent global states
   Using Chandy's global-state-detection algorithm, assume that once a process receives a 'marker' message, it saves its global state and does not send markers to the other processes. Would would the impact be?
   The global state that was recorded could be an inconsistent global state. If process P sends a marker to process Q while a message is in transit from $Q \rightarrow P$, then Q receives that marker, the state would be recorded as P not having received any message, but Q having sent the message. There would also be no record of anything on the channel between P and Q.
5. Chandy's global-state-detection algorithm and strongly consistent global states
   Is there a way to use Chandy's global state detection algorithm to record a strongly consistent global state? What would the benefit be of doing so?
   Yes, you could simply use the existing algorithm multiple times, and only record finish once we get a global state that has no messages on any channel. This may take arbitrarily long, and never finish, but the benefit would be that the state that is recorded is a state the system actually passed through. As Chandy's algorithm is now, it may record a global state that the system was never actually in.
6. Huang's termination-detection algorithm and message loss
   Why is that Huang's termination-detection algorithm assumes no message loss (what would happen if messages were lost)?
   Huang's algorithm attaches weights to each message sent between processes. When the controller process has all the weight and it is idle, the system is in a termination state. If a message were to be lost during Huang's algorithm, weight would bleed from the system, and the algorithm would never be able to detect termination.
7. Cristian's external clock synchronization and clock drift rates
   $\boldsymbol{U_{min} = min(1 + \rho)}$. Why? What is the effect of increasing $\boldsymbol{\rho}$?
   If $U < \min(1 + \rho)$ then there is no way to get a reading fast enough to sync the clock. Since ρ is defined as the drift rate of the clock, the message cannot possibly be received within the minimum one-way delay skewed by the fast running clock. In other words, if a clock has a drift rate of 0.1, and the minimum delay is 0.1, there is no possible way for a message to get from the sender to the receiver in < 0.11 seconds, since it takes 0.1

seconds for the message to physically get there, but the clock is skewed by a factor of 0.1, so by the sender's clock, it takes **at least** 0.11 seconds for a message to physically get to the receiver.

8. Berkeley internal clock synchronization and drift of entire system

   Say a set of processes $\{P, Q, R, S\}$ are synchronized internally using the Berkeley algorithm. Let the clocks in processes $\{P, Q, R\}$ have the same drift value, ρ=0. What would be the effect on the time of the entire system if the clock in S had such a high drift, that $\Delta_{PD} > \gamma$ at every read? Specify any assumptions you make.

   The system clock would always be accurate. Since all $|\Delta| > \gamma$ clock differences are dropped when calculating the correction values, none of the processes $\{P, Q, R\}$ would change with any update. Assuming the clock value in the system was accurate to the real time within some value δ, the system clock would always be within δ of the real time.

9. Network Time Protocol and clock accuracy among different stratums

   Why does the clock accuracy decrease as the stratum number of the NTP client increases?

   Since every stratum level increase means that the NTP client is one level further away from the reference clock (usually GPS). As in, stratum 1 is connected directly to a GPS device to get the exact time, stratum 2 gets its time updates from a stratum 1 server. Since there is message delay between the stratum 1 and 2 servers, some inaccuracy is introduced when going from stratum 1 to stratum 2. This inaccuracy is compounded when going from stratum 2 to 3, and so on from 3 to 4.

10. Lamport's mutual exclusion algorithm and message traffic

    The message complexity of Lamport's mutual exclusion is $3(N-1)$, because each of the N processes sends 3 types of messages. What are these types and why are they necessary? Is there any way to decrease this message complexity?

    The three types are REQUEST, ACK, and RELEASE. The REQUEST message is necessary to let other processes know the resource is needed. The ACK is needed to show that a process is able to acquire the resource. The RELEASE is needed to signify that a process is done, and another process can acquire the resource. There is a way to decrease the message complexity. If the process priority is determined beforehand, every process can send an ACK to begin with, and if the resource is not needed then a RELEASE can be sent immediately, followed by an ACK to the next process in the priority list. This method would reduce the message complexity to $2(N-1)$, but if the resources aren't needed very much then it would create lots of constant traffic for no purpose.

11. Ricart's mutual exclusion algorithm and deadlock

    In Ricart's algorithm, is message loss tolerated? If so, how? If not, why not?

    No. In the case of message loss, it will cause a deadlock. If either a REQUEST or REPLY message is lost, at some point the sequence numbers will pass the sequence number in the lost message. When that happens, the process that was relying on the lost message (the sender for a REPLY or the receiver for a REQUEST) will not grant a REPLY to any requests that come in.

12. Ricart's mutual exclusion algorithm and starvation

    Starvation in mutual exclusion algorithms is sometimes a desired quality. For instance, if one of the processes is a very low priority, it may be useful to starve that process for an arbitrary amount of time. Is there a way to do this using Ricart's algorithm? If so, how? If not, what in the algorithm prevents this?

    Yes. Since Ricart's algorithm includes sequence numbers, it is possible to set sequence number increment values to very high numbers, so that a low priority gets to enter its critical section much less often.

13. Ricart's mutual exclusion algorithm and message traffic

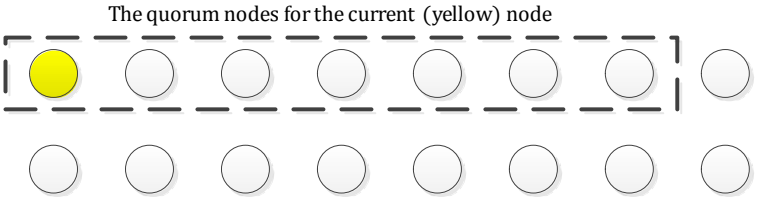
    What is the message complexity of Ricart's distributed mutual exclusion algorithm? Is there any way to improve on this?

    The message complexity is $2(N-1)$, because there are $N-1$ REQUEST messages and $N-1$ REPLY messages. This can improved to $N-1 \leq M \leq 3(N-1)$ using an implicit

REPLY algorithm, or N using a broadcast or ring based algorithm.

14. Maekawa's mutual exclusion algorithm and violation of mutual exclusion

    Maekawa's quorum based mutual exclusion algorithm uses a grid based approach to select the quorum members for any specific node. What would happen if instead of a grid, the nodes were arranged in a line, and a process chose $2\sqrt{N} - 1$, starting with itself, nodes to be its quorum members instead of using a grid? i.e.

    Violation of mutual exclusion. There could be a case where two nodes do not share any two quorum members, so they would both enter the CS at the same time.

15. Maekawa's mutual exclusion algorithm and starvation

    How is starvation prevented in Maekawa's algorithm?

    Every quorum node keeps a queue of received requests, and once it is not locked, will reply in the order the requests are received. Also, since very node has at least one quorum node in common with every other quorum node, it is guaranteed that the locks will be granted in the order that the requests are received.

16. Maekawa's mutual exclusion algorithm and message traffic

    In Maekawa's quorum based mutual exclusion algorithm, why is it that $2\sqrt{N} - 1$ requests are sent, but only $\sqrt{N}$ responses are needed to enter the critical section?

    Since Maekawa's algorithm uses a grid based selection method to select the quorum members for each node, it is guaranteed that there are no two nodes that have $> \sqrt{N} - 1$ distinct quorum members. This way, once you have $\sqrt{N}$ responses, you are guaranteed that no other node is in the CS.

17. Raymond's mutual exclusion algorithm and violation of mutual exclusion

    How could a node misbehave that would allow a violation of mutual exclusion?

    If a node sends a token reply when it doesn't hold the token, it will cause a violation of mutual exclusion. This will essentially create two tokens in the system, one at the previous token holder, and one at the new token holder. This also means that from this point on, mutual exclusion is broken.

18. Raymond's mutual exclusion algorithm and deadlock

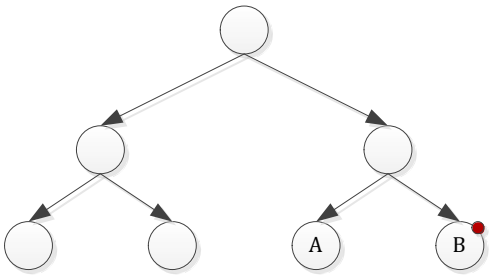
    Raymond's algorithm assumes a tree structure to the system of nodes. To be resilient to network failures, Professor R suggests using a ring structure in the network. How would this effect the algorithm.

    It could cause deadlock. Since a REQUEST message could be sent to a neighbor to the "left" immediately before receiving a TOKEN message from the "right." If that's the case, the TOKEN message will follow the REQUEST message around the ring indefinitely, and no node will be able to enter the CS.

19. Raymond's mutual exclusion algorithm and starvation

    What would the effect of using a stack instead of a queue in Raymond's algorithm to represent the awaiting requests?

    In a system where the nodes are arranged in a tree, using a stack instead of a queue could lead to indefinite starvation. For example,

    If node B has the token, and all other nodes but A in the system are waiting for the

token, then node A requests the token, A will receive it next. If during the time A has the token, B requests it, B will be given the token next. Since all other nodes are waiting for the token, they will not make any requests for the token, and every request A and B make will be granted immediately. Effectively, A and B will starve the rest of the system.

20. Raymond's mutual exclusion algorithm and message traffic

Raymond's algorithm is a token passing protocol for achieving mutual exclusion. What is the worst case for the message complexity of this algorithm (in big-O notation), and why?

This worst case for the message complexity is $O(N)$. If all the nodes are arranged in a straight line, a request from end of the line to a holder at the opposite end must pass N messages to the holder, and N messages for the token to reach the requester. So the complexity is $2(N-1)$ or $O(N)$.