Distributed Algorithms: Study Questions

Paul Gribelyuk (pg1312, a5)

May 17, 2013

1 Exam 2010

1.1 1a

The basic routing problem is that processes are not generally fully connected and messages would need to be directed along links. The problem is of finding efficient links to send messages along so they arrive quickly at destination.

1.2 1b

Destination-based routing is the construction of routes based on the destination (usually via routing tables), whereas source-based routing is used in broadcast and multicast to propagate messages along (based on source, being the root of the tree). Dijsktra's algorithm is not useful here. The Floyd Warshall algorithm computes shortest path between any two pairs of nodes in a graph.

1.3 1c

1.3.1 i

This algorithm is a way of measuring the distance between a specific node v and any other node in the network for the purpose of destination-based routing. It works by having each process update the minimum distance between itself and the node v by receiving the message of another processor with a known distance, and then seeing if the weight w_{ij} added to that distance is lower than it's own estimate of the distance. As this propagates through the network, nodes will contain information about how best to get to node v.

1.3.2 ii

The total number of messages is initially 2 for v. If there are N total processors in the ring, it will take $N/2$ rounds before each processor knows the minimum distance. The 2nd round has 2 processors receiving messages and sending 4 messages (2 to v). The 3rd round, the 2 new processors will send 4 messages to neighbors. In general, there will always be 4 messages per round so total message count is $2N-2$.

1.4 2a

The coordination problem for two processes is the act of making sure we have these properties in the algorithm:

- Validity - the coordinator chosen should be from among the proposed ones

- Termination - all processes involved in deciding a coordinator must eventually decide
- Agreement - the coordinator was chosen unanimously

1.5 2b

The problem as I stated it is ambiguous as I have not specified the failure model. Under a no crash model, the problem is trivially solvable. If the channel is faulty, then it is unsolvable because neither process can depend on the receipt of its last message, so this follows as a proof by contradiction.

1.6 2c

The coordination problem, as stated above, applies to synchronous models. If we change the model to asynchronous, then this would keep the problem unsolvable without making stronger assumptions.

1.7 2d

To solve leader-election with 1 link failure in a ring, we know that when there are no failures, it takes N rounds in unidirecitonal LCR since the leader waits to hear his own UID. If there is a link failure, we have to assume bidirectional links and each process proceeds to send messages further and further (they have to return back to the sender with some value) away and if it receives it from the opposite direction (then the ring is complete and he is the leader), or it comes back twice with the same length travelled and it's own UID (in which case, there is a link failure but at the end of the link, it sends back both the length travelled and the max over all UIDs).

1.8 3a

1.8.1 i

Two-phase commit is where all processes send votes to a coordinator and the coordinator chooses whether to commit or not based on unanimity of the collected votes.

1.8.2 ii

The centralized 2PC relies on a coordinator, while the decentralized 2PC has messages being sent to everyone at each stage. Thus, if all processes receive "yes" from all other processes, the send "preCommit" messages around, and once they receive those from everyone, they will go through with the operation. Centralized 2PC requires $\Theta(N)$ messages, whereas decentralized requires $\Theta(N^2)$ messages. Time complexity is the same.

1.9 3b

1.9.1 i

Three types of fairness are:

- Strong fairness: if a resource is requested infinitely often, it will be granted infinitely often

- Weak fairness: if a resource is requested continuously after a certain point, it will be granted infinitely often thereafter
- No fairness: the only way a resource may be granted is if there is no other alternative. No other guarantee can be made

1.9.2 ii

Peterson 2P supports Weak Fairness as the setting of the “turn” variable places the process in the queue, continously.

1.10 3c

1.10.1 i

Failure detectors are necessary for leader election and consensus problems, as failed processes will not be considered for voting.

1.10.2 ii

An eventually perfect detector will no suspect any correct process after some specified time. An eventually strong detector will not suspect at least one correct processes after some specified time.

1.11 4a

1.11.1 i

Two events are *concurrent* if there is no way from either process (where each event occured) to reach the other process via message passing. By looking at the history of message sends and receives, it is possible to determine whether any events occured at the other process. If neither process receives or sends messages before the events, then they have no further information by which to order them.

1.11.2 ii

All events concurrent with q2: p1, p2, p3

1.11.3 iii

The total time order relation is defined in terms of messages passing between processes. Each message has a logical clock counter for the sending process, which is the largest number of clocks of which it is aware of via messages sent or received, or inter-process events occuring. If a process receives a larger value than its internal value, it uses it going forward, else, it uses its own. In the case that two processes have the same logical clock, their PID will break the tie.

1.11.4 iv

For vector clocks, each process sends a vector of all clock values it is aware of for each other process (including itself). The receiver updates personal vector of clocks, taking the latest one to reflect most up-to-date information. For a vector timestamp to be before another:

$$V_a < V_b \Rightarrow \forall k \quad V_a[k] \leq V_b[k] \quad and \quad \exists j \quad V_a[j] \neq V_b[j]$$

Then if $a \to b$ implies either a and b are in the same process, in which case $V_a < V_b$ trivially, or a is a send and b is receipt of the same message, in which case, this is also trivial. Alternatively, if $V_a < V_b$, need to show that $a \to b$. Prove by contradiction: suppose a does not cause b, then a is a send but b is not the receive. The receipt of the message sent at a must be after b so $V_b[i] < V_a[i]$ by definition, not knowing about the send. Alternatively, if b is a receive from a different point prior to a, then it won't know about the upated value of $V_a[i]$ when a happens and again $V_b[i] < V_a[i]$.

1.12 4b

1.12.1 i

Processes share resource and communicate via messages asynchronously:

- Safety - no two processes have access to resource at the same time
- Order - requests granted in order made
- Liveness - every request to access resource eventually satisfied

1.12.2 ii

The centralized solution is to have a resource allocator process receiving messages requesting access for critical region. Access is granted via unique reply to the first receipt of message. The rest are held in a queue. Thus, for the coordinator, if $b \to b'$ then the sender of b gets access first.

1.12.3 iii

This solves safety and liveness. However, requests cannot be granted in order made.

2 Exam 2012

2.1 1a

A distributed algorithm is one which performs computations across a variety of machines without any guarantee of shared resources, having to rely on message passing for communication. A centralized algorithm can be significantly simpler as all states about which it needs to know are readily available and observable.

2.2 1b

2.2.1 i

The consensus algorithm must satisfy:

- Validity: the result is from one of the ones proposed
- Agreement: all processes must come to the same conclusion
- Termination: all processes must eventually vote

2.2.2 ii

As an example, if a process can failstop, then the algorithm will block on step 3. receiving values from all other processes. Also, if a process with the minimum value failstops in the middle of sending a value, then some processes will have the correct minimum value while others will have an inferior value, thus failing on Agreement

2.2.3 iii

If F processes crash, then we need $F + 1$ rounds, so that after that many rounds, we are guaranteed to have a round without failures and then all broadcast values will indeed be correctly distributed to everyone.

2.3 2a

2.3.1 i

Lamport's algorithm iteratively sends UM(n,m) messages to all processes not yet sent (as seen via the path down the tree, that the algorithm has taken). The algorithm isn't solvable for $N = 3$ and $F = 1$. Now if $N < 3F + 1$, then the 3 parts of N subdivided evently can be represented as one part each, among them having 1 failed subset. This cannot be solved.

2.3.2 ii

The loyal generals take v_{def} when there is a tie in the calculation of majority. v_{def} is sent when the faulty process doesn't send anything.

2.3.3 iii

For $N = 100$, need $F = 33$ to reach consensus, taking 34 rounds.

2.4 2b

2.4.1 i

Constructing message table involves making 1 round for the general to send the messages to liutenants and one round of sharing the data (since $F = 1$). Message table for process i will have values $(V, [1, i])$.

2.4.2 ii

The decision reached will be the decision at the leaf nodes in Stage 3.

2.4.3 iii

In total, $(n-1)+(n-1)\cdot(n-2) = 16$ are sent.

2.5 3a

2.5.1 i

One idea to compute AND on a ring: first elect a leader, which takes $\Theta(n)$. Then have the leader send his value in one direction only and wait to receive the AND value after n rounds, which also takes $\Theta(n)$.
Alternatively, nodes with 0 send values and terminate. Nodes with 1 wait for $n/2$ rounds forwarding any messages and halting at 0 if they receive any. Else they halt at 1.

2.5.2 ii

Worst case complexity is when there are mostly 1s

2.5.3 iii

The worst case is worse than the best case because in the best case, the algorithm could terminate immediately

2.5.4 iv

Time complexity is on average $n/4$

2.6 2b

2.6.1 i

Asynchronous AND on a ring??? WTF!!!

2.6.2 ii

no clue about message complexity

2.7 4

Very similar to Exam 2010 question 1

3 Exam 2011

3.1 1a

A *synchronous* network has processes which know the upper bound for the amount of time any process requires as well as an upper bound on message delay. Physical clocks at synchronous network nodes count time perfectly in unison. Alternatively an *asynchronous* network cannot put any restrictions on computation of message delay times. Physical clocks in asynchronous networks are not perfect and subject to disparities.

3.2 1b

The Leader Election problem is the problem of selecting a process from the network according to a prespecified criteria. The conditions are:

- Termination: All correct processes vote at some point
- Consistency: All correct processes decide on the same process

3.3 1c

The FloodMax algorithm assumes that processes know the diameter (maximum distance to any other node) of the network.

3.3.1 i

Pseudocode for FloodMax:

3.3.2 ii

The time complexity is D, the diameter of the network

3.3.3 iii

The message complexity is $D \times N$ where N is the number of nodes in the network. Each round sees N messages being sent and there are D such rounds.

```
Data: $i$ is current processor ID
$D$ is the diameter of the network
$Max \leftarrow i$
$Stat \leftarrow$ Unknown for $r = 1$ to $D$ do
    send [VAL: $Max$]
    receive [VAL: v]
    if $v > Max$ then
        $Max \leftarrow v$
    end
end
if $Max == i$ then
    $Stat \leftarrow$ Leader
else
    $Stat \leftarrow$ Follower
end
```

3.3.4 iv

The *FloodMaxOpt* algorithm reduces message traffic by not sending another message if its internal value for Max has not been updated, since that message would not provide any further information to the receiving nodes.