Distibuted Algorithms Exam Study Notes

Paul Gribelyuk (pg1312, a5)

April 9, 2013

1 Given Notes

1.1 Introduction slides

- Characterizing good models: *accuracy* (how true are the properties compared to real world) vs *tractability* (can be analyzed)
- Modeling costs of an algorithm: state size, complexity, amount of communication
- Modelling elements:
  - *process* is a computational resource
  - *channel* is a communication link between processes; used to transport messages
- Communication Operations:
  - send: sends message m "atomically" from P_i to P_j; invoked by sender
  - receive: invoked by receiver and is a *state change* for receiver
  - broadcast: a combination of send operations to all processes; not atomic and not deterministic
- Problems in Distributed Algorithms: reaching a decision (consensus on the same value or decision about a process being different, i.e. a leader), detecting remote failures, tolerating process or communication failures, measuring progress
- Keywords for consensus:
  - *Agreement*: all processes decided the same value
  - *Termination*: all processes make a decision in finite time
  - *Validity*: the decided value must be one that was proposed
- With a lossy channel, agreement is impossible, even among 2 processes. To prove this, use contradiction, assume it is possible, and notice that the sender of the last message in the communication sequence cannot rely on it being delivered, and thus, his actions cannot depend on the contents of that message. Therefore, that message is not necessary, therefore, the optimal sequence of messages is shorter. Iterative argument shows there is no optimal sequence of messages
- Failure types:
  - *Failstop* A process fails by stopping (halting) and remains stopped. This can be detected externally
  - *Crash* Same as failstop by other processes may not be able to detect the failure

  - *Crash+Link* Needs further explanation
  - *Receive Omission* Process does not receive all messages
  - *Send Omission* Process does not send all messages it wants to
  - *General Omission* Combination of receive and send omissions
  - *Byzantine* Failure by doing random behavior
- Channels:
  - *Reliable*: A message is received if the sending process has not failed at tiem of send and receiving process has not failed at time of arrival
  - *Quasi-reliable*: A message is received if both sending and receiving processes have not failed prior to arrival of message
  - *Unreliable*: There is no guarantee of a sent message being received
- 3 Properties of Failure-free networks:
  - *Liveness* - each process will execute another step eventually
  - *Safety* - messages are received exactly as many times as they are sent
  - *Liveness*(again) - receipt of a message is guaranteed if process will take infinitely many steps
- Robust algorithms assume permanent process failures and use failure detection
- Stabilizing algorithms assume processes will be eventually correct and work with transient failures, i.e. non-permament
- Requirements for decision problems: termination and consistency (all decide the same value for either leader or consensus)
- Reliable Atomic broadcast means all processes receive their messages in correct order as they were sent; equivalent to solving consensus
- Types of distributed networks:
  - Synchronous: there are known upper bounds on computation delays and message delays and processes have synchronized clocks; this allows them to be able to estimate how long each "round" should take and if they don't receive a message from another process, they know for certain that it has failed. Solving election is easy: each process with $PID = i$ waits $i - 1$ rounds and receives messages. If it receives a message before round i, then it decides the leader to be the sender of that message. Otherwise, it broadcasts its own UID in round i.
  - Asynchronous: Unbounded execution time, unbounded message sending/receiving time, unsynchronized clocks. In this case, leader election is hard

1.2 Synchronous Algorithms

We use the synchronous network model of definite upper bounds on processing and communication delays with synchronized clocks. The network model is one of a graph $G = (V, E)$ of verteces and edges.

- $distance(i, j)$ is the shortest path from i to j
- $diameter(G) = max_{i,j} distance(i, j)$

A process is a state machine with following properties:

- *states*
- *start* (initial states)
- *msg* function to generate messages (function from state space to output message)
- *trans* is transition function from state to state (function from state space together with message space to state space)

A channel can hold a message from message space M or null (no message).
Synchronous execution happens in *rounds* with the steps:

1 generate outgoing message using *msg* at each process

2 use *trans* at each process to obtain new state (using current state and incoming message).

Communication failures resulting in message omission are modeled as null messages in channels.
A synchronous execution is an infinite sequence:

$$C_0, M_1, N_1, M_2, N_2, C_2, \ldots$$

where:

- C_r is state at round r
- M_r is messages sent at round r
- N_r is messages received at round r

Identical processes arranged in a ring network cannot solve leader election (proof by contradiction: if exists and execution sequence leading to election, then all processes reach the decision they are the leader at the same time), therefore we need UID to distinguish them (Lynch p.27)

1.2.1 Leader Election In a Ring

LCR algorithm

- unidirectional communication
- processes don't know size of ring
- each process has unique UID

Algorithm:

- Each process forwards UID to neighbor
- Each process discards received UID if it's less that their own
- Each process forwards UID if received UID is greater than their own
- If process receives their own UID, then they're the leader

Proving correctness:

1 The *snd* function of a process sending values to process r hops away from max-UID-valued process is u_{max} for any r. Thus, if $r = n - 1$, then i_{max} receives its own UID and declares itself the leader

2 No other process sees their own UID because i_{max} acts as a barrier to forwarding messages (since it has the highest UID).

3 Algorithm terminates by the leader sending a "halt" message around.

Analysis:

- Time Complexity: $2 \cdot n$ if counting n rounds to send "halt" message around
- Communication Complexity: $\frac{n(n-1)}{2} + n \approx O(n^2)$

1.2.2 Leader Election in General Network

FloodMax solves leader election on any connected graph by asuming processes know upper bound on diameter (D_{max}) of network. Algorithm:

- Each process maintains maximum UID seen
- Each process broadcasts maximum on all available channels
- If after D_{max} rounds, a process has same maximum seen as its own UID, then it is the leader.

So, each process keeps track of its own UID, the max seen UID, the stat ($\{leader, unknown, follower\}$), and number of rounds. *msg* function places max seen on all output channels so long as number of rounds less than D_{max}. *trans* function updates number of rounds by 1, receives messages from all input channels, updates max seen and makes decision about whether it should update its stat variable (initially set to unknown).
Analysis:

- Need to prove correctness
- Time complexity is D_{max} rounds
- Communication complexity is $D_{max} \cdot |E|$ sice $|E|$ messages get sent in each round.

Another approach: *FloodMaxOpt* cuts down on message complexity by only forwarding the max seen if it has been updated by incoming values. Need to keep another boolean value to keep track of that.

1.2.3 Breadth-first search in General Network

SyncBFS
Setup: processes unaware of size of network and laid out in general graph.
Problem is to find the *spanning tree* starting from a specific node i_0

- Each process receives probe messages on incoming channels.
- If that process is unmarked, it marks itself and declares one of the senders of the probe messages its parent
- If a process marked itself in the previous round, then it sends a probe on all outgoing channels

Analysis:

- Time complexity: number of rounds
- Communication Complexity: $O(|E|)$

To account for processes knowing their children (as well as parents), allow for back-channel replies from processes to which probe messages are sent ("child" or "not child").