And now for something completely different...

RANDOMIZATION

Randomized Algorithms

Extend transition function to accept as input

- a random number
- from a bounded range
- under some fixed distribution

Why is it important?

The bad news:
- randomization alone does not generally affect
  - impossibility results
  - leader election in anonymous network still impossible!
  - worst case bounds

The good news:
- randomization + weakening of problem statement does

Example: Randomized Leader Election

- Impossibility in anonymous rings still holds
- but can now elect a leader with some probability
- So weaken LE as follows

Safety: In every configuration of every admissible execution, at most one processor is in an elected state

Liveness: At least one processor is elected with some non-zero probability

Behaviors allowed by weakened specification:
- terminate without a leader
- never terminate
A first result

Assume
- synchronous ring
- non-uniform ring
- processor can randomly choose identifiers

Theorem
There is a randomized algorithm which, with probability \( c > \frac{1}{e} \), elects a leader in a synchronous ring; the algorithm sends \( O(n^2) \) messages

The Algorithm

**Code for processor \( p_i \)**

```plaintext
Initially
0: pid, := 0 with probability \( 1/2^n \)
1: send pid, to left

2: upon receiving \( \langle S \rangle \) from right
3: if \( |S| = n \) then
4: if pid, is unique max(S) then
5: elected := true
6: else
7: elected := false
8: else
9: send \( \langle S \rangle \mid pid, \rangle \) to left
```

**Observations:**
- randomization used once
- one execution for each element of \( \{1,2\}^n \)

Analysis

What is the probability that the algorithm terminates with a leader?

\[
\Pr[\text{termination}] \geq \frac{1}{e} - \frac{1}{n}
\]

Message Complexity:

\( O(n^2) \)

Not good enough?

Trade off more time and messages for higher probability of success
- if \( |S| = n \) and \( p_i \) detects no single max in \( S \)
  - choose new \( pid, \)
  - restart algorithm
- becomes a set of \( n \)-tuples
each of which is a possibly infinite sequence over \( \{1,2\} \)
Analysis

Probability of success in iteration $k$

$$\left(1-c\right)^{k+1} - c$$

Time complexity:

- worst-case number of iterations: \[E[T] = \sum_{1}^{\infty} P(E[T = t])\]
- expected number of iterations: \[E[T] = \sum_{1}^{\infty} P(T = t)\]

Expected value of $T$:

Expected message complexity: $O(n^2)$

Impossibility of Uniform Algorithms

Theorem

There is no uniform randomized algorithm for leader election in a synchronous anonymous ring that terminates in even a single execution for a single ring size.

Summary

- No deterministic solution for anonymous rings
- No solution for uniform anonymous rings (even when using randomization)
- Protocols with $O(n^2)$ and $O(n \log n)$ messages for uniform rings
- $\Theta(n \log n)$ lower bound on message complexity for practical protocols
- $O(n)$ message complexity for uniform synchronous rings
Atomic Commit

The objective
Preserve data consistency for distributed transactions in the presence of failures

Model
• For each distributed transaction $T$:
  – one coordinator
  – a set of participants
• Coordinator knows participants, but participants don’t necessarily know each other
• Each process has access to a Distributed Transaction Log (DT Log) on stable storage

The setup
• Each process $p_i$ has an input value $vote_i$:
  $vote_i \in \{\text{Yes, No}\}$
• Each process $p_i$ has an output value $decision_i$:
  $decision_i \in \{\text{Commit, Abort}\}$
AC Specification

- **AC-1:** All processes that reach a decision reach the same one.
- **AC-2:** A process cannot reverse its decision after it has reached one.
- **AC-3:** The Commit decision can only be reached if all processes vote Yes.
- **AC-4:** If there are no failures and all processes vote Yes, then the decision will be Commit.
- **AC-5:** If all failures are repaired and there are no more failures, then all processes will eventually decide.

Comments

- **AC1:**
  - We do not require that all processes reach a decision
  - We do not require that all correct processes reach a decision (with link failures, this is equivalent to the “2 Generals Problem”)
- **AC4:**
  - Avoids triviality
  - Allows Abort even if all processes have voted yes

NOTE: A process that does not vote Yes can unilaterally abort

To be or not to be…

**Uncertainty:** A process is uncertain if it has voted Yes but does not have sufficient information to commit

What if a process becomes isolated while uncertain?

Uncertainty + Failures can cause *blocking:*

a process is blocked if it must wait for the recovery of another process before deciding

Independent Recovery?

**Independent Recovery:** the ability of a recovering process to reach a decision without communicating with other processes.

No independent recovery + total failures \[\rightarrow\] blocking!
A few character-building facts

**Proposition 1**: If communication failures or total failures are possible, then every AC protocol may cause processes to become blocked

**Proposition 2**: No AC Protocol can guarantee independent recovery of failed processes

Notes on 2PC

- Satisfies AC-1 to AC-4
- But not AC-5

A process may be waiting for a message that may never arrive

Use **Timeout Actions**

No guarantee that a recovered process will reach a decision consistent with that of other processes

Allow processes to write data to their DT-Log

2-Phase Commit

Nico Gazardo, IBM for the Italian Social Security System (1971)
Rediscovered and published: Lampson, Sturgis (1976)

1. Coordinator \( c \) sends VOTE-REQ to all participants.
2. When participant \( p_i \) receives a VOTE-REQ, it responds by sending a vote to the coordinator.
   - If vote, \( = \) NO, then decide, \( \Rightarrow \) ABORT and \( p_i \) halts.
3. Coordinator \( c \) collects votes from all:
   - If all votes are yes, then decide, \( \Rightarrow \) COMMIT; sends COMMIT to all
   - Else decide, \( \Rightarrow \) ABORT; sends ABORT to all participants who voted YES
   - \( c \) halts.
4. If participant \( p_i \) receives COMMIT then decide, \( \Rightarrow \) COMMIT
   - Else decide, \( \Rightarrow \) ABORT
   - \( p_i \) halts.

Timeout Actions

Processes are waiting at the beginning of steps 2, 3, and 4

**Step 2** A participant is waiting for VOTE-REQ from coordinator

Since this happens before vote, participant can decide ABORT and halt.

**Step 3** Coordinator is waiting for vote from participants

Coordinator can decide ABORT, send ABORT to all participants which voted YES, and halt.

**Step 4** Participant (who has voted YES) waits for COMMIT or ABORT

Participant cannot decide: it must run a **Termination Protocol**
Termination Protocols

Simple TP
Wait for coordinator to recover

Cooperative TP
Ask other participants

- Works, because coordinator is never uncertain
- Process may be blocked unnecessarily
- Use Cooperative TP!

Cooperative Termination Protocol

- When an uncertain process times out, it sends a DECISION-REQ message to all other participants
- Let $p$ be the initiator, $q$ be a responder

1. if $q$ has decided, then $q$ sends its decision value to $p$
   $p$ decides accordingly
2. if $q$ has not yet voted then decide$_q := \text{ABORT}$
   $q$ sends ABORT to $p$
3. if $q$ is uncertain then $q$ cannot help $p$

Cooperative
Termination Protocol

1. when coordinator sends VOTE-REQ, it writes START-2PC to its DT Log.
2. when participant is ready to vote YES, writes YES to DT Log before sending YES to coordinator (writes also list of participants).
3. when participant is ready to vote NO, it writes ABORT to DT Log.
4. after participant receives decision value, it writes it to DT Log.

Specify which information should be written to DT Log

Logging Actions

1. When coordinator sends VOTE-REQ, it writes START-2PC to its DT Log.
2. When participant is ready to vote YES, writes YES to DT Log before sending YES to coordinator (writes also list of participants).
3. When participant is ready to vote NO, it writes ABORT to DT Log.
4. When coordinator is ready to decide COMMIT, it writes COMMIT to DT Log before sending COMMIT to participants.
5. After participant receives decision value, it writes it to DT Log.

What happens during recovery?

1. if DT Log contains START-2PC, then process is coordinator:
   - if it also contains ABORT or COMMIT, no problem
   - if it contains nothing else, ABORT
2. otherwise, process is participant:
   - if DT Log contains COMMIT or ABORT, no problem
   - if it contains no YES vote, unilaterally decide ABORT
   - if it contains YES but no decision value, use termination protocol