Circumventing Impossibility

Partial Synchrony

Circumventing Impossibility

• Consensus is an important building block for fault-tolerant computing
  – Universal: any deterministic fault-tolerant service can be implemented on top of it
• Yet, it is impossible in very practical environments
  – Asynchronous systems
  – Are they really practical?
Circumventing Impossibility

- Key observation: most practical settings are never completely asynchronous
  - We could expect interleaving, arbitrarily long periods of synchrony and asynchrony
- Synchrony assumptions:
  - Ways to formally capture types of semi-synchronous behavior found in practice
  - Implementability of Consensus under various assumptions

Sources of timing uncertainty

<table>
<thead>
<tr>
<th></th>
<th>Relative process speeds</th>
<th>Message/access delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message passing</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Shared memory with variables</td>
<td>Y</td>
<td>NA</td>
</tr>
<tr>
<td>Shared memory with objects</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Synchrony Assumptions

• Real time clock
  – At each tick of the clock some processes take exactly one step of their protocol

• Bounded relative process speeds:
  – ∃ integer Φ>0: in any time interval in which some process takes Φ real time steps, each correct process takes at least 1 step

• Bounded message delay:
  – ∃ integer Δ>0: if p sends m to q at time t, then q receives m by the time t+Δ

More assumptions

• Messages are received in the order which respects the real time order of their send events
• Atomic broadcast is available
• Atomic receive/send
Dolev, Dwork and Stockmeyer, “On Minimal Synchronism Needed for Distributed Consensus”

<table>
<thead>
<tr>
<th>mb pc</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>n</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>n</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>n</td>
<td>0</td>
<td>1</td>
<td>n</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>n</td>
<td>n</td>
<td>0</td>
<td>0</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

Crash failures
s=0

Partial Synchrony

- Φ and Δ
  - Processes (communication) are (is) partially synchronous if Φ (Δ) holds eventually (◊)
    - Synchronous if Φ (Δ) holds always
  - Φ (Δ) holds eventually
    - There exists a Global Stabilization Time (GST) such that Φ (Δ) holds in [GST,∞)

Dwork, Lynch and Stockmeyer, Consensus in the Presence of Partial Synchrony
Models of Partial Synchrony

Summary of the DLS Results

<table>
<thead>
<tr>
<th>Failure type</th>
<th>Synch</th>
<th>Asynch</th>
<th>◊Δ,◊Φ</th>
<th>◊Δ,◊Φ</th>
<th>Δ,◊Φ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash</td>
<td>t</td>
<td>∞</td>
<td>2t+1</td>
<td>2t+1</td>
<td>t</td>
</tr>
<tr>
<td>Omission</td>
<td>t</td>
<td>∞</td>
<td>2t+1</td>
<td>2t+1</td>
<td>2t+1</td>
</tr>
<tr>
<td>Auth. Byz</td>
<td>t'</td>
<td>∞</td>
<td>3t+1</td>
<td>3t+1</td>
<td>2t+1</td>
</tr>
<tr>
<td>Byz</td>
<td>3t+1</td>
<td>∞</td>
<td>3t+1</td>
<td>3t+1</td>
<td>3t+1</td>
</tr>
</tbody>
</table>

All bounds are tight
System Components

- Consensus Algorithm
- Round Simulation
- Partially Synchronous Environment with failures

Round Simulation
(Basic Round Model)

- Abstracts away timeliness assumptions
  - The failure models stay the same
  - 4 Consensus algorithms, 64 round simulations
- Processing is divided into rounds
- Each round consists of
  - Send sub-round
  - Receive sub-round
  - Computation sub-round
The round structure

- Send sub-round:
  - Each process sends messages to any subset of the processes

- Receive sub-round:
  - Some subset of the messages sent to the process during the send sub-round are delivered

- Computation sub-round:
  - Each process executes a state transition based on the set of messages just received

Requirements

- There is a round GST such that
  - All messages sent from correct processes to correct processes at $r \geq GST$ are delivered during $r$

- Processes do not know when GST occurs
Crash and Omission failures

- n processes: $p_1, \ldots, p_n$
- $n/2$ resilient Consensus
- NU Agreement, Strong Unanimity and Termination

The protocol structure

Phase 1
- Round 1
- Round 2
- Round 3
- Round 4
- ...

Phase $k$
- Round $4k-3$
- Round $4k-2$
- Round $4k-1$
- Round $4k$
- ...

Phase $k$ is coordinated by a process $p_i$: $i \equiv k \mod n$
Phase $k \equiv i \mod n$

- $p_j$: send $(\text{list}, k)$ to $p_i$, where
  - list = $\{v\}$, if $v$ is the only locked $v \in V$
  - list = $V$, if no values are locked
  - list = $\emptyset$, otherwise

- $p_i$: $w$ is in lists of $\geq n-t$ processes
  - Send $(\text{lock}, w, k)$ to all processes

- $p_j$: receives $(\text{lock}, w, k)$
  - Lock $w$ (unlocks previous locks for $w$),
  - send $(\text{ack}, k)$ to $p_i$

- $p_i$: receives $(\text{ack}, k)$ from $t+1$ processes:
  - Decide $w$, but does not halt

Phase $k \equiv i \mod n$

- Round 4 of $k$: Lock-release
- $p_j$: broadcasts $(v, h)$ for each $v$ such that $v$ was locked by $p_j$ at phase $h$
- $p_j$: receives $(w, h')$ from some process:
  - If $p_j$ locked $(v, h)$ with $v \neq w$ and $h \leq h'$, unlock $(v, h)$
Agreement

- Let $k$ be the smallest phase at which some process decides
  - $p_i, i=k \mod n$ decides $v$
- at least $t+1$ processes locked $v$ at phase $k$
- it is impossible for any further coordinator to lock a different value since any two sets of sizes $n-t$ and $t+1$ intersect

Validity

- Very weak validity is satisfied
  - More than a single decision is possible
- Achieving weak (strong) unanimity is a simple exercise
  - And is left as such 😃
Termination

• After GST all processes learn about the highest phase value locked by any process (if any) \( \Rightarrow \) at most one value \( v \) is locked by all correct processes
• All processes will send to the coord. either \( v \) or the entire set \( V \) (which includes \( v \))
• The coordinator will see some value appearing \( \geq n-t \) times, etc…

Authenticated Byzantine

• A simple modification of the algorithm:
  – Every message is signed
  – Proposals have a sequence of \( n-t \) signed messages attached as a proof
  – Everybody verifies proofs, signatures
Impossibility for $2 \leq n \leq 2t$

- Partition $n$ processes into two sets each of which is of size at least 1 and at most $t$
- Initialize each set with conflicting values
- Fail either set to force conflicting decisions in two different executions
- Combine these two executions to achieve an execution with conflicting decisions