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

	Relative process speeds	Message/access delay
Message passing	Y	Y
Shared memory with variables	Y	NA
Shared memory with objects	Y	Y

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 $\Phi>0$: 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 $\Delta>0$: if p sends m to q at time t, then q receives m by the time $t+\Delta$

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”

pc \ mb	00	01	11	10	00	01	11	10
00	0	0	n	0	0	0	n	0
01	0	0	n	0	1	n	n	1
11	n	n	n	n	n	n	n	n
10	0	0	n	n	0	0	n	n
	s=0				s=1			

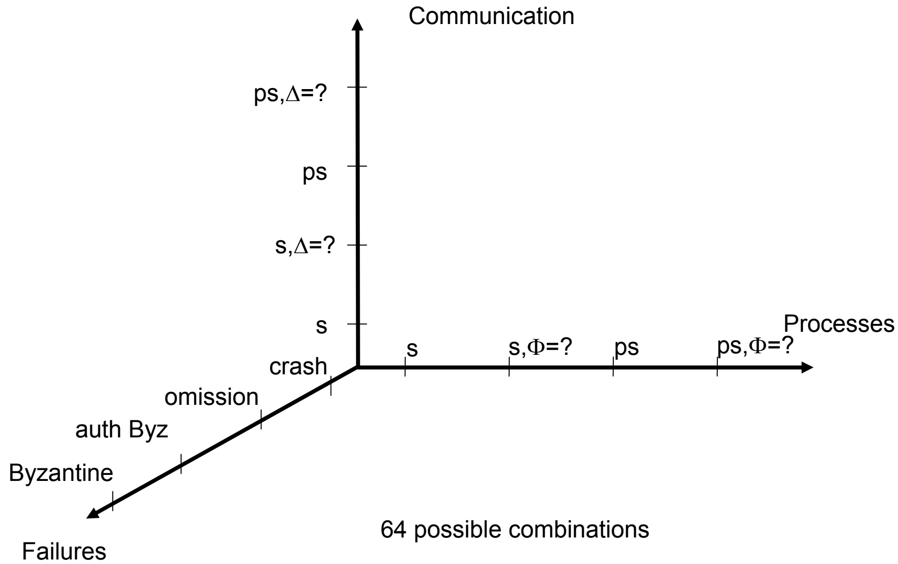
Crash failures

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

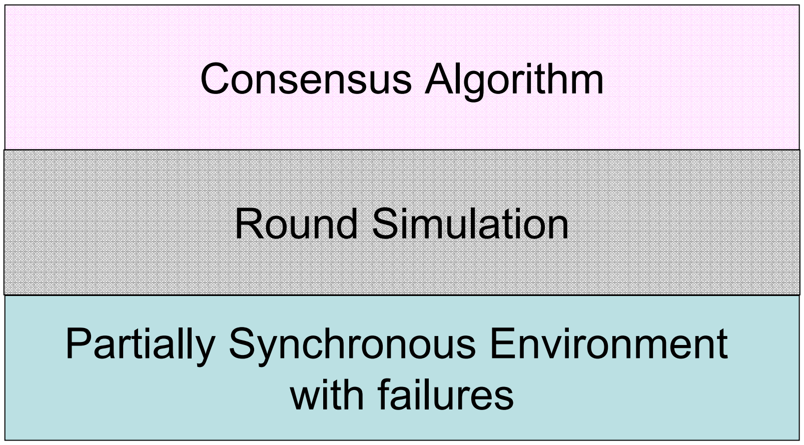


Summary of the DLS Results

Failure type	Synch	Asynch	◊Δ,Φ	◊Δ, ◊Φ	Δ, ◊Φ
Crash	t	∞	2t+1	2t+1	t
Omission	t	∞	2t+1	2t+1	2t+1
Auth. Byz	t^*	∞	3t+1	3t+1	2t+1
Byz.	3t+1	∞	3t+1	3t+1	3t+1

ALL BOUNDS ARE TIGHT

System Components



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: $i \equiv k \bmod n$

Phase k $\equiv$ i mod n

- p_j: send (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 (lock, w, k) to all processes
- p_j: receives (lock, w, k)
  - Lock w (ulocks previous locks for w),
  - send (ack, k) to p_i
- p_i: receives (ack, k) from t+1 processes:
  - Decide w, but does not halt

round 1 of k

round 2 of k

round 3 of k

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≠w and h≤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) ➔ 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