Consensus in the presence of partial synchrony

Dwork, C., Lynch, N., & Stockmeyer, L. (1988).
Journal of the ACM, Vol. 35, No. 2, pp. 288-323

Adding time to the model

Consensus problem

Initially each processor p_i has a value v_i drawn from some domain V of values

- The correct processors must all decide on the same value, and if their initial values are all the same, say v, then v must be the common decision — Safety
- Each correct processor should eventually make a decision — Livenes

The consensus protocol should operate correctly if some of the processors are faulty

Consensus problem

Protocols to achieve agreement in synchronous distributed system are well-known

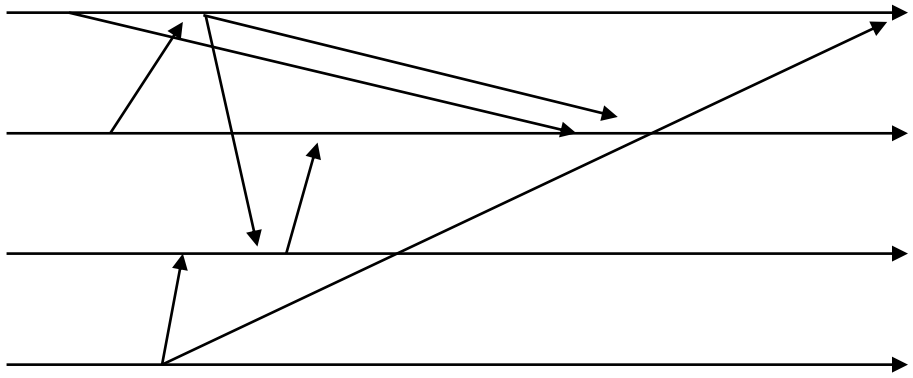
Synchrony does not hold in practice

Agreement in an asynchronous system is impossible

Asynchronous and synchronous models

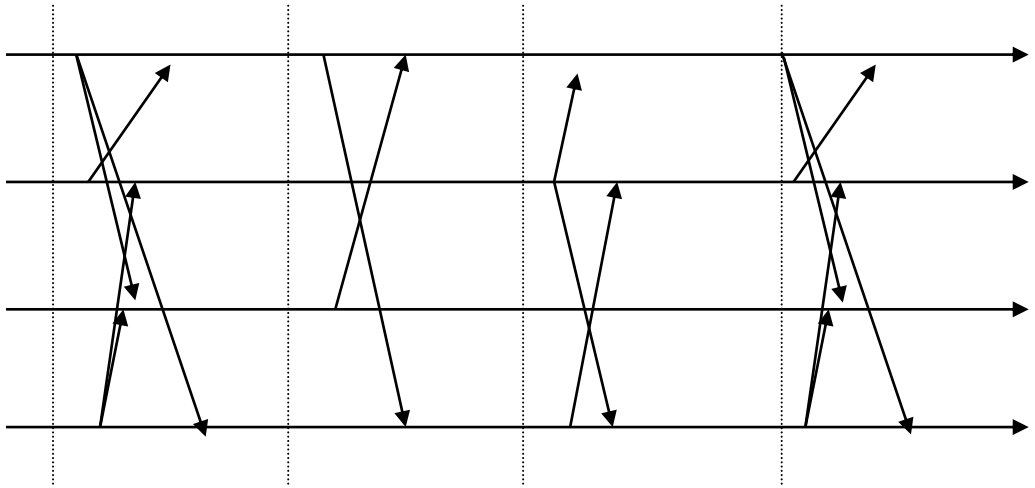
Asynchronous

- No assumptions about process speed
- Network can arbitrarily delay a message
- No timing assumptions whatsoever



Synchronous

- Assume that every process will run within bounded delay
- Assume that every link has bounded delay
- Usually described as “synchronous rounds”



Asynchronous and synchronous models

Asynchronous model is too weak

- overly pessimistic,
- real systems have clocks,
- “most” timing meets expectations …
- but heavy tails

Synchronous model is too strong

- real systems lack a way to implement synchronized rounds

Consensus problem

Solution: partially synchronous model

Real distributed systems (e.g., Internet):

- synchronous?
- asynchronous?
- partially synchronous? (eventually temporary stable)

Partial synchrony

Timing assumptions

Delay Δ for message transmission

Some processors are Φ times faster than others

Partial synchrony

Two versions of partial synchrony:

1. Fixed bounds Δ and Φ exist, but they are not known a priori
2. The bounds Δ and Φ are known, but are only guaranteed to hold starting at some unknown time GST (global stabilization time)

Separating the correctness conditions

We require an algorithm to satisfy the safety conditions no matter how asynchronously the system behaves, that is, even if the bounds Δ and Φ do not hold eventually

We only require termination in case the bounds hold eventually

Maximum resiliency possible

Our results determine precisely the maximum resiliency possible with partially synchronous communication and processors:

- For crash faults t-resilient consensus is possible iff $N \geq 2t + 1$.
- For Byzantine faults t-resilient consensus is possible iff $N \geq 3t + 1$.

Consensus and synchrony

Smallest number of processors N for which a t-resilient consensus protocol exists

Failure type	Synchronous	Asynchronous	Partially synchronous communication and synchronous processors	Partially synchronous communication and processors	Partially synchronous processors and synchronous communication
Fail-stop	t	∞	$2t + 1$	$2t + 1$	t
Omission	t	∞	$2t + 1$	$2t + 1$	$[2t, 2t + 1]$
Authenticated Byzantine	t	∞	$3t + 1$	$3t + 1$	$2t + 1$
Byzantine	$3t + 1$	∞	$3t + 1$	$3t + 1$	$3t + 1$

Models of partial synchrony

$N \geq 2,\ t \geq 1$

Correctness of a consensus protocol

For any set C containing at least N - t processors and any run R in which the processors in C are correct and the behavior of the processors not in C is allowed by the fault type F, the protocol achieves:

- Consistency. No two different processors in C decide differently.
- Unanimity. If all initial values are v and if any processor in C decides, then it decides v.
- Termination. If R is infinite, then every processor in C makes a decision.

Common method

Our protocols use variations on a common method:

- A processor p_i tries to get other processors to change to some value v that p has found to be “acceptable”;
- p_i decides v if it receives sufficiently many acknowledgments from others that they have changed their value to v, so that a value different from v will never be found acceptable at a later time

Rotating coordinator

Basic round model

Message exchange in synchronous rounds, divided in three subrounds:

- send: every processor can send messages to any subset of the processors
- receive: some subset of the messages sent in previous subround is received
- computation: state transition based on the received messages

Assumptions:

- There is some round GST such that all messages that are sent at or after GST arrive in the same round they were sent
- Messages can be lost before GST, this is not considered as faulty behavior

Protocols in the basic model

1. Fail-stop and omission faults ($N \geq 2t+1$)
2. Bizantine faults with authentication ($N \geq 3t+1$)
3. Bizantine faults without authentication ($N \geq 3t+1$)

Protocols in the basic model

Every processor p_i stores

- some initial value v out of a value domain V
- PROPER $\subseteq$ V, set of proper values
- locks on some values
- a message buffer

Proper values

- if there is exactly one initial value, this is the only proper one
- otherwise, every $v \in V$ is proper

Fail-stop and omission failures

Initially, each processor's set PROPER contains just its own initial value

Each processor attaches its PROPER set to every message it sends

Whenever a processor p receives a PROPER set from another processor that contains a particular value v, then p puts v into its own PROPER set

Fail-stop and omission failures

The rounds are organized into alternating trying and lock-release phases

- trying: 3 rounds
- lock-release 1 round

Each pair of corresponding phases is assigned an integer, starting with 1

We say that phase k belongs to processor p_i if $k = i \bmod N$

(p_i is the processor that can propose a value in phase k)

Fail-stop and omission failures

At various times during the algorithm, a processor may lock a value v

Locks on values have an associated phase number:

p_j has lock on value v for phase k $\Rightarrow$ p_j thinks p_i might decide v at phase k

A value v is acceptable to p if p does not have a lock on any value except possibly v

Round 1 (of phase k)

- Each processor sends a list of all its acceptable values that are also in its PROPER set to processor p_i
- Processor p_i attempts to choose a value to propose in next round:
  - In order for processor p_i to propose v, it must have heard that at least N - t processors find value v acceptable and proper at the beginning of phase k

Round 2

- Processor p_i proposes a value v: broadcast message (lock v, k)
- Processor p_j receiving a (lock v, k) message, locks v associating the phase number k with the lock (and releases any earlier lock on v)

Round 3

- Every processor that locked v sends an acknowledged message to p_i
- p_i decides v if it receives at least t + 1 acknowledgements

Round 4

- Every processor broadcasts on which values it has locks
- A processor releases a lock if another processor has a lock on a different value in a more recent phase

Lemmas

1. It is impossible for two distinct values to acquire locks with the same associated phase
2. Suppose that some processor decides v at phase k, and k is the smallest numbered phase at which a decision is made; then at least t+1 processors lock v at phase k; moreover, each of the processors that locks v at phase k will, from that time onward, always have a lock on v with associated phase number at least k
3. Immediately after any lock-release phase that occurs at or after GST, the set of values locked by correct processors contains at most one value

Theorem

- Assume the basic model with fail-stop or omission faults
- Assume $N \geq 2t + 1$

- Then Algorithm 1 achieves consistency, strong unanimity, and termination for an arbitrary value domain

Protocols in partial synchrony models

Each of the three models of partial synchrony can be used to simulate the basic model

Partially synchronous communication and synchronous processors

Partially synchronous communication and processors

Partially synchronous processors and synchronous communication

Partially synchronous communication

$\Phi=1$, Δ holds but is unknown:

- round r in synchronous algorithm is simulated by N + r steps
- first N steps: send messages as previously
- last r steps: receive messages
- each round gets a unique identifier, attached to every sent message (easy: processors share a common clock)
- messages from earlier rounds are ignored

$\Rightarrow$ as soon as $r \geq \Delta$ holds, messages are delivered in the same round

Partially synchronous communication

Φ=1, Δ holds eventually:

- one round in synchronous algorithm is simulated by N + Δ steps
- first N steps: send messages as previously
- last Δ steps: receive messages
- each round gets a unique identifier, attached to every sent message (easy: processors share a common clock)
- messages from earlier rounds are ignored

$\Rightarrow$ as soon as Δ holds, message are delivered in the same round

$\Rightarrow$ after agreement is reached, Δ can stop holding

Partially synchronous communication and processors

Distributed clocks (protocol)

- Processors have to agree on a (approximately) common notion of time
- Global clock simulated by private (software) clocks
- Processors exchange ticks and claims about ticks
- Private clocks are increased if there are enough valid claims to do so

Fault-tolerant variation on the logical clocks defined by Lamport

Partially synchronous communication and processors

Once we have defined the distributed clocks, the protocols of the partially synchronous communication model are simulated by letting each processor use its private clock to determine which round it is in

Several “ticks” of each private clock are used for the simulation of each round in the basic model

The distributed clocks (protocol) creates a lot of overhead

Consensus and partial synchrony

All protocols proposed:

- reach agreement in polynomial time
- have optimal resiliency

But:

- specification of the protocols could be more exact
- protocols create a lot of overhead
- practical application questionable

Consensus is possible in partially synchronous models!

Consensus in the presence of partial synchrony

Dwork, C., Lynch, N., & Stockmeyer, L. (1988).
Journal of the ACM, Vol. 35, No. 2, pp. 288-323

Adding time to the model

Byzantine faults with authentication

Protocol is divided into (numbered) phases with two subphases

- trying: 3 rounds
- lock-release 1 round

Each processor attaches its PROPER set and its initial value to every (signed) message it sends

In phase k, processor p_i with i = k mod N is the processor to propose a value

At various times during the algorithm, a processor may lock a value v

Locks on values have an associated phase number

If p_j has lock on value v for phase k $\Rightarrow$ p_j thinks p_i might decide v at phase k

Round 1 (of phase k)

- Each processor sends a list of all its acceptable values that are also in its PROPER set to every other processor
- If p_j receives claims from at least t + 1 other processors that a value v is in their PROPER sets, it adds v to PROPER
- If p_j receives initial values from more than 2t + 1 processors among which there are not t + 1 equal values, p_j sets PROPER = V

- Processor p_i attempts to choose a value to propose:
  - In order for processor p_i to propose v, it must have heard that at least N - t processors find value v acceptable and proper at phase k

Round 2

- Processor p_i proposes a value v (proof of acceptability included: set of signed messages)
- Processor p_j receiving a valid proposal locks v associating the phase number, earlier locks on v are released

Round 3

- Every processor that locked v sends an acknowledged message to p_i
- p_i decides v if it receives more than 2t + 1 replies

Round 4

- Every processor broadcasts on which values it has locks (on phase k)
- A processors releases a lock if another processor has a lock on a different value

Comments

- It is impossible for two distinct values to acquire valid locks at the same trying phase if that phase belongs to a correct processor
- Suppose that some correct processor decides v at phase k, and k is the smallest numbered phase at which a decision is made by a correct processor; then at least t + 1 correct processors lock v at phase k; moreover, each of the correct processors that locks v at phase k will, from that time onward, always have a lock on v with associated phase number at least k
- Immediately after any lock-release phase that occurs at or after GST, the set of values locked by correct processors contains at most one value