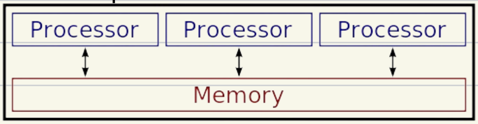
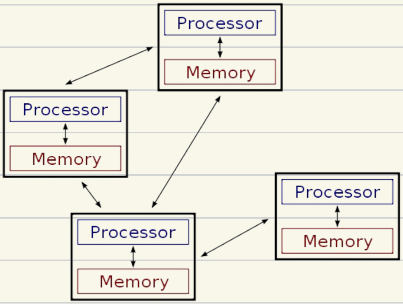
Distributed Systems

software system in which networked computer components communicate and coordinate actions by passing messages



Parallel systems - shared memory

Distributed systems - no shared components



Inherent Distribution:

- Information dissemination
- Distributed process control
- Cooperative work
- Distributed storage

Distribution:

- Performance
- Scalability
- Availability
- Fault tolerance

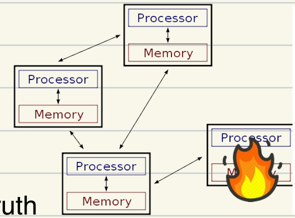
Characteristics:

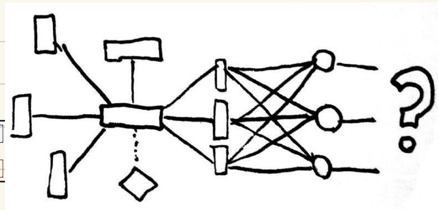
- Computational entities with own memory (synchronise distributed state)
- Communication with message passing
- Each entity storing part of the complete picture
- Failure tolerant

Fallacies:

- reliable network
- no latency
- infinite bandwidth
- constant topology
- no transport cost
- homogeneous network

4 Main Problems





- Partial failures
- Unreliable networks
- Unreliable time
- No single source of truth

Synchronous system - bounded execution speed / message delivery times

-> timed failure detection

-> time based coordination

-> worst case performance

Asynchronous system - no assumptions about execution speed

- request lost
- remote node down
- response lost

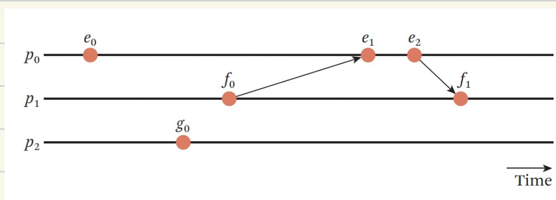
-> timeouts(upper boundary) + retry request - partially-synchronous

Logical Time

causality based clock

Strict partial order:

- Irreflexivity - items not comparable with self
- Transitivity: a<b, b<c, a<c
- Antisymmetry: a<=b, b<=a -> a=b



Strict total order - a<=b or b<=a or a=b

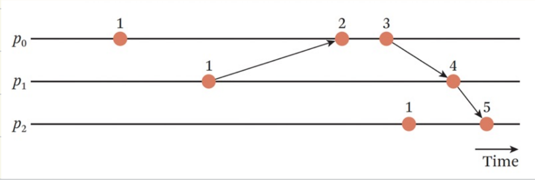
Events:

- process perform local computation
- process send a message
- process receive a message

a -> b

- a,b - events in same node, then a occurs before b
- a - send message, b - receive message
- transitive

Lamport Timestamps



each process p maintains a counter LT(p)

process p perform action -> LT(p)+1

process p send message + LT(p)

process p reserve message from q; LT(p) = max(LT(p), LT(q)) + 1

E0.	[A(0), B(0), C(0), D(0)]
E1. A sends to C	[A(1), B(0), C(0), D(0)]
E2. C receives from A	[A(1), B(0), C(2), D(0)]
E3. C sends to A	[A(1), B(0), C(3), D(0)]
E4. A receives from C	[A(4), B(0), C(3), D(0)]
E5. B sends to D	[A(4), B(1), C(3), D(0)]
E6. D receives from B	[A(5), B(1), C(3), D(2)]

LT(E6) < LT(E4) but doesn’t mean E6 -> E4 (E4,E6) - independent

Vector Clock

system of N nodes, each node i has vector Vi of size N

Vi[i] - number of events occured at node i

Vi[j] -number of events node i knows occurred at node j

local event - Vi[i]++

i sends message to j + Vi

j receives Vi, update Vj[a] = max(Vi[a], Vj[a])

Vi = Vj iff Vi[k] = Vj[k] for all k

Vi <= Vj iff Vi[k] <= Vj[k] for all k

a -> b => VC(a) < VC(b)

VC(a) < VC(b) => a->b

E0.	[A(0,0,0,0), B(0,0,0,0), C(0,0,0,0), D(0,0,0,0)]
E1. A sends to C	[A(1,0,0,0), B(0,0,0,0), C(0,0,0,0), D(0,0,0,0)]
E2. C receives from A	[A(1,0,0,0), B(0,0,0,0), C(1,0,1,0), D(0,0,0,0)]
E3. C sends to A	[A(1,0,0,0), B(0,0,0,0), C(1,0,2,0), D(0,0,0,0)]
E4. A receives from C	[A(2,0,2,0), B(0,0,0,0), C(1,0,2,0), D(0,0,0,0)]
E5. B sends to D	[A(2,0,2,0), B(0,1,0,0), C(1,0,2,0), D(0,0,0,0)]
E6. D receives from B	[A(2,0,2,0), B(0,1,0,0), C(1,0,2,0), D(0,1,0,1)]

Consensus

agreement in presence of faulty node

- Resource allocation
- Committing a transaction
- Synchronising state machines
- Leader election
- Atomic broadcast

Fault tolerant consensus:

- Byzantine - 3f+1 nodes with f traitors
- Crash - 2f+1 nodes with f traitors

Crash fault-tolerant consensus protocol:

- Safety - never return incorrect result, non-Byzantine conditions
- Availability - answer if n/2 + 1 servers are operational
- No clocks - no RTC dependency
- Immune to stranglers - n/2 + 1 servers vote, result is safe

Panos consensus algorithm:

Proposer - chooses a value, sends to set of acceptors to collect

Acceptor - vote to accept or reject value

Learner - adopt value (when majority accepts)

Voting:

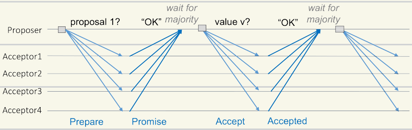
proposer selects proposal number n and sends “prepare” request to acceptors

n - higher than every previous proposal -> Promise -> ignore future proposals <n

accept - send previous proposal number and corresponding accepted value

Replication:

proposer - response from majority -> accept request

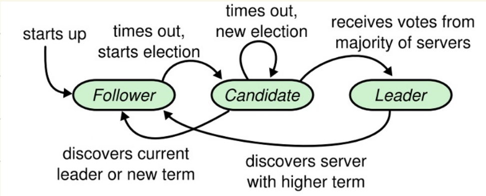


Raft:

Leader election:

choose leader

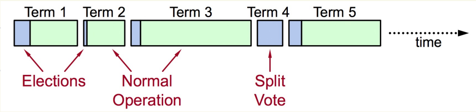
crash -> new leader



Log replication:

leader accepts commands, appends to log

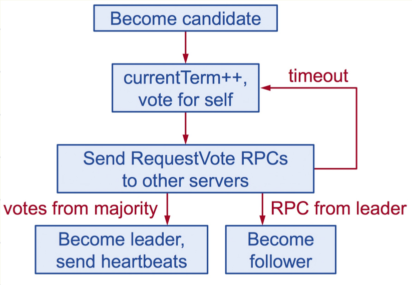
leader replicates log to other servers



Candidate (for leader): ask for votes

Leader: accept log entries, replicated to servers

Follower: replicate leader’s state machine



log entry - index position in log

leader sends AppendEntries

log entry committed - replicated to majority