Consensus in Cloud Computing and Paxos

Dr. Rajiv Misra
Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss the ’consensus problem’ and its variants, its solvability under different failure models .
- We will also discuss the industry use of consensus using ‘Paxos’.

Common issues in consensus

Consider a group of servers attempting together:

- Make sure that all of them receive the same updates in the same order as each other **[Reliable Multicast]**
- To keep their own local lists where they know about each other, and when anyone leaves or fails, everyone is updated simultaneously **[Membership/Failure Detection]**
- Elect a leader among them, and let everyone in the group know about it **[Leader Election]**
- To ensure mutually exclusive (one process at a time only) access to a critical resource like a file **[Mutual Exclusion]**

So what is common?

- All of these were groups of processes attempting to ***coordinate*** with each other and reach ***agreement*** on the value of something
  - The ordering of messages
  - The up/down status of a suspected failed process
  - Who the leader is
  - Who has access to the critical resource
- All of these are related to the ***Consensus*** problem

What is a Consensus Problem?

Formal problem statement:

- N processes
- Each process p has

  input variable xp : initially either 0 or 1

  output variable yp : initially b (can be changed only once)
- **Consensus problem:** design a protocol so that at the end, either:
  1. All processes set their output variables to 0 (all-0's)
  2. Or All processes set their output variables to 1 (all-1's)

What is Consensus? (2)

- Every process contributes a value
- ***Goal is to have all processes decide same (some) value***
  - Decision once made can't be changed
- There might be other constraints:
  - **Validity:** if everyone proposes same value, then that's what's decided
  - **Integrity** : decided value must have been proposed by some process
  - **Non-triviality** : there is at least one initial system state that leads to each of the all-0's or all-1's outcomes

Why is it Important?

- Many problems in distributed systems are **equivalent to (or harder than)** consensus!
  - Perfect Failure Detection
  - Leader election (select exactly one leader, and every alive process knows about it)
  - Agreement (harder than consensus)
- So consensus is a very important problem, and solving it would be really useful!
- **So, is there a solution to Consensus?**

Two Different Models of Distributed Systems

- **Different Models of Distributed Systems:**
  (i) Synchronous System Model and
  (ii) **Asynchronous System Model**

(i) Synchronous Distributed System

- Each message is received within bounded time
- Drift of each process' local clock has a known bound
- Each step in a process takes lb < time < ub

E.g., A collection of processors connected by a communication bus, e.g., a Cray supercomputer or a multicore machine

Asynchronous System Model

(ii) Asynchronous Distributed System

- No bounds on process execution
- The drift rate of a clock is arbitrary
- No bounds on message transmission delays

E.g., The Internet is an asynchronous distributed system, so are ad-hoc and sensor networks

- This is a more **general (and thus challenging)** model than the synchronous system model.

- A protocol for an asynchronous system will also work for a synchronous system (but not vice-versa)

Possible or Not

- **In the synchronous system model:**
  - **Consensus is solvable**
- **In the asynchronous system model:**
  - **Consensus is impossible to solve**
  - Whatever protocol/algorithm you suggest, there is always a worst-case possible execution (with failures and message delays) that prevents the system from reaching consensus
  - **Powerful result (*FLP proof*)**
  - Subsequently, **safe or probabilistic solutions** have become quite popular to consensus or related problems.

FLP

- One of the most important results in distributed systems theory was published in April 1985 by **Fischer, Lynch and Patterson (FLP)**

- Their short paper 'Impossibility of Distributed Consensus with One Faulty Process', which won the Dijkstra award, placed an upper bound on what it is possible to achieve with distributed processes in an asynchronous environment.

Let's Try to Solve Consensus!

System model (Assumptions):

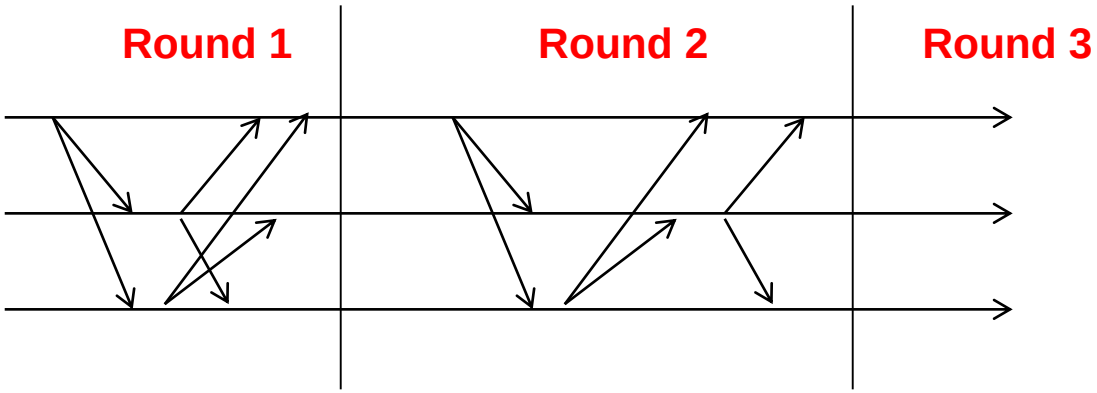
- **Synchronous system:** bounds on
  - Message delays
  - Upper bound on clock drift rates
  - Max time for each process step

  e.g., multiprocessor (common clock across processors)

- **Processes can fail by stopping (crash-stop or crash failures)**

Consensus in Synchronous Systems

- **For a system with at most *f* processes crashing**
  - All processes are synchronized and operate in **"rounds"** of time.
  - The algorithm proceeds in *f+1* rounds (with timeout), using reliable communication to all members
  - $Values^r_i$ *:* the set of proposed values known to p_i at the beginning of round *r*.



Consensus in Synchronous System

Possible to achieve!

- For a system with at most *f* processes crashing
  - All processes are synchronized and operate in "rounds" of time
  - the algorithm proceeds in *f+1* rounds (with timeout), using reliable communication to all members.
  - $Values^r_i$: the set of proposed values known to p_i at the beginning of round *r*.

- Initially $Values^0_i = \{\}$; $Values^1_i = \{v_i\}$

for round = 1 to *f+1* do

multicast ($Values^r_i - Values^{r-1}_i$) // iterate through processes, send each a message

$Values^{r+1}_i \leftarrow Values^r_i$

for each V_j received

$Values^{r+1}_i = Values^{r+1}_i \cup V_j$

end

end

d_i = **minimum**($Values^{f+1}_i$) // consistent minimum based on say, id (not minimum value)

Why does the Algorithm work?

- After *f+1* rounds, all non-faulty processes would have received the same set of Values. Proof by contradiction.
- Assume that two non-faulty processes, say p_i and p_j, differ in their final set of values (i.e., after *f+1* rounds)
- Assume that p_i possesses a value *v* that p_j does not possess.
  - p_i must have received *v* in the **very last** round
    - Else, p_i would have sent *v* to p_j in that last round
  - So, in the last round: a third process, p_k, must have sent *v* to p_i, but then crashed before sending *v* to p_j.
  - Similarly, a fourth process sending *v* in the **last-but-one round** must have crashed; otherwise, both p_k and p_j should have received *v*.
  - Proceeding in this way, we infer at least one (unique) crash in each of the preceding rounds.
  - This means a total of *f+1* crashes, while we have assumed at most *f* crashes can occur => contradiction.

Consensus in an Asynchronous System

- **Impossible to achieve!**

- Proved in a now-famous result by **Fischer, Lynch and Patterson, 1983 (FLP)**
  - Stopped many distributed system designers dead in their tracks
  - A lot of claims of "reliability" vanished overnight

Recall

Asynchronous system: All message delays and processing delays can be arbitrarily long or short.

Consensus:

- Each process p has a state
  - program counter, registers, stack, local variables
  - input register xp : initially either 0 or 1
  - output register yp : initially b (undecided)
- **Consensus Problem:** design a protocol so that either
  - all processes set their output variables to 0 (all-0's)
  - Or all processes set their output variables to 1 (all-1's)
  - Non-triviality: at least one initial system state leads to each of the above two outcomes

Consensus Problem

- Consensus **impossible** to solve in asynchronous systems **(FLP Proof)**
  - **Key to the Proof:** It is impossible to distinguish a failed process from one that is just very very slow. Hence the rest of the alive processes may stay forever when it comes to deciding.

- But Consensus important since it maps to many important distributed computing problems.

Paxos

Paxos algorithm:

- Most popular "**consensus-solving**" algorithm
- Does not solve consensus problem (which would be impossible, because we already proved that)
- But provides **safety** and **eventual liveness.**
- A lot of systems use it
  - Zookeeper (Yahoo!), Google Chubby, and many other companies

Paxos

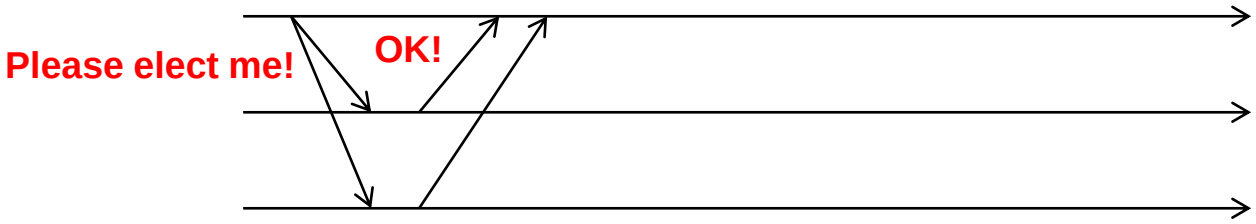
- Paxos is invented by **Leslie Lamport**
- Paxos provides **safety** and **eventual liveness**
  - **Safety:** Consensus is not violated
  - **Eventual Liveness:** If things go well sometime in the future (messages, failures, etc.), there is a good chance consensus will be reached. But there is no guarantee.
- FLP result still applies: Paxos is not ***guaranteed*** to reach Consensus (ever, or within any bounded time)

Paxos Algorithm

- Paxos has **rounds**; each round has a unique ballot id
- Rounds are asynchronous
  - Time synchronization not required
  - If you're in round *j* and hear a message from round *j+1*, abort everything and move over to round *j+1*
  - Use timeouts; may be pessimistic
- Each round itself broken into phases (which are also asynchronous)
  - Phase 1: A leader is elected (**Election**)
  - Phase 2: Leader proposes a value, processes ack
  - Phase 3: Leader multicasts final value

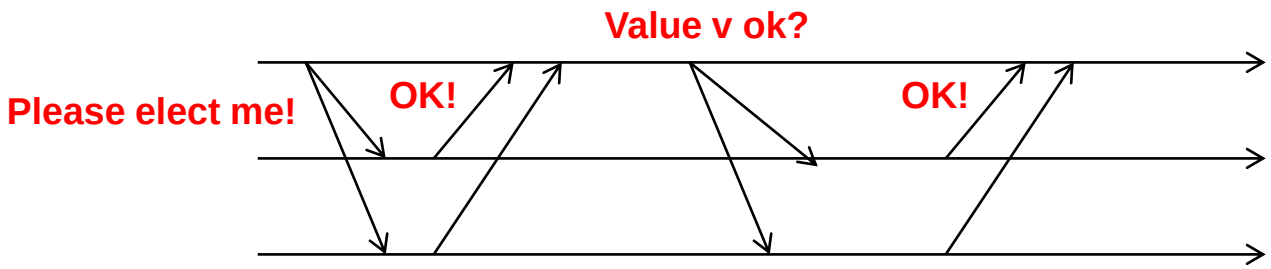
Phase 1: Election

- Potential leader chooses a unique ballot id, higher than seen anything so far
- Sends to all processes
- Processes wait, respond once to highest ballot id
  - If potential leader sees a higher ballot id, it can't be a leader
  - Paxos tolerant to multiple leaders, but we'll only discuss 1 leader case
  - Processes also **log** received ballot ID on disk
- If a process has in a previous round decided on a value v', it includes value v' in its response
- If **majority (i.e., quorum)** respond OK then you are the leader
  - If no one has majority, start new round
- (If things go right) A round cannot have two leaders (why?)



Phase 2: Proposal

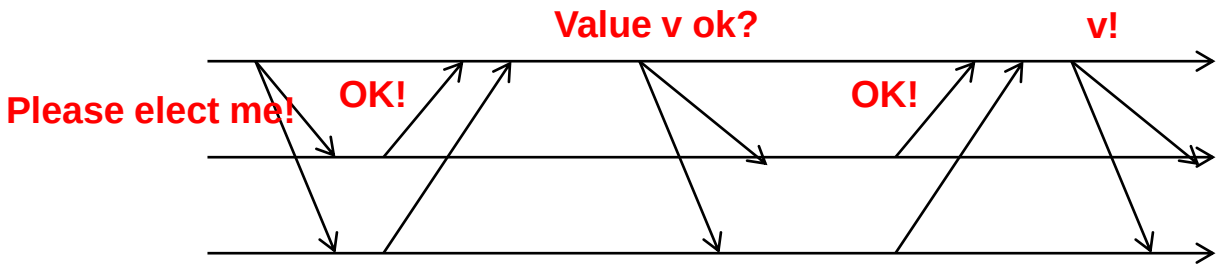
- Leader sends proposed value v to all
  - use v=v' if some process already decided in a previous round and sent you its decided value v'
  - If multiple such v' received, use latest one
- Recipient logs on disk; responds OK



Phase 3: Decision

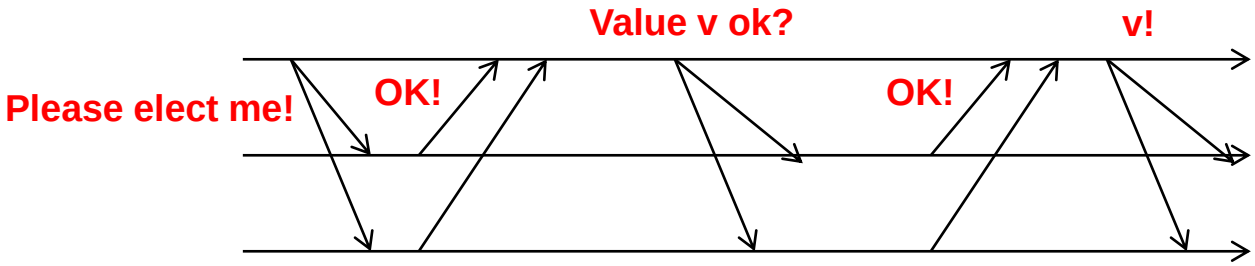
- If leader hears a **majority** of OKs, it lets everyone know of the decision

- Recipients receive decision, log it on disk



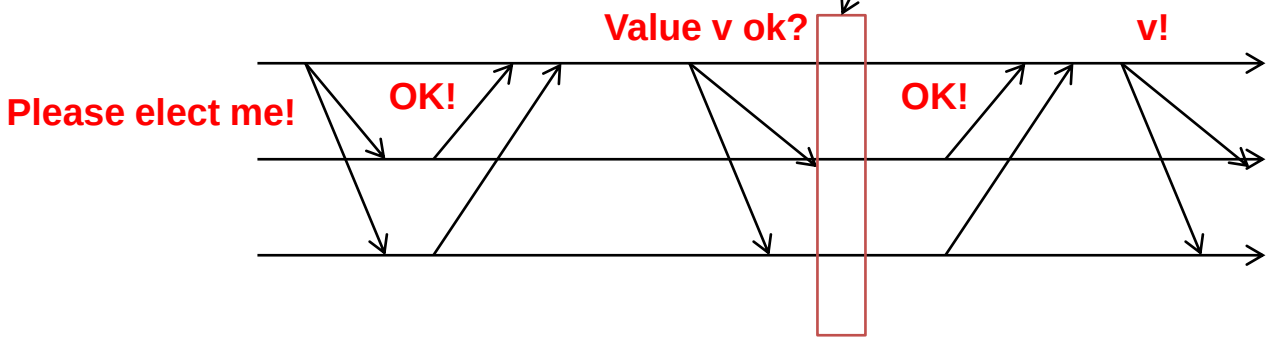
Which is the point of No-Return?

- That is, when is consensus reached in the system



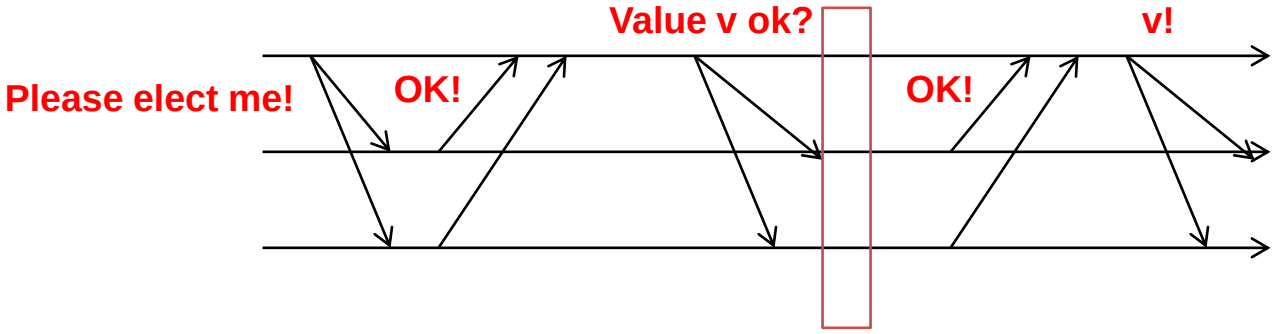
Which is the point of No-Return?

- If/when a majority of processes hear proposed value and accept it (i.e., are about to/have respond(ed) with an OK!)
- Processes ***may not know it yet,*** but a decision has been made for the group
  - Even leader does not know it yet
- What if leader fails after that?
  - Keep having rounds until some round completes



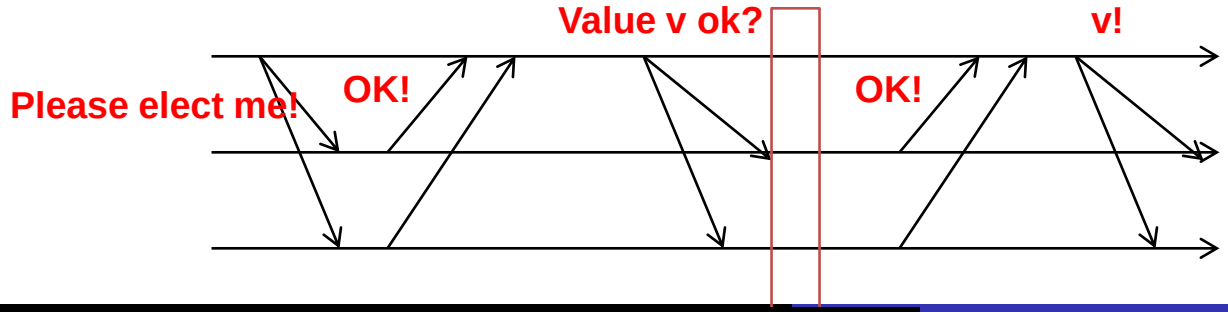
Safety

- If some round has a majority (i.e., quorum) hearing proposed value v' and accepting it, then subsequently at each round either: 1) the round chooses v' as decision or 2) the round fails
- **Proof:**
  - Potential leader waits for majority of OKs in Phase 1
  - At least one will contain v' (because two majorities or quorums always intersect)
  - It will choose to send out v' in Phase 2
- Success requires a majority, and any two majority sets intersect



What could go Wrong?

- **Process fails**
  - Majority does not include it
  - When process restarts, it uses log to retrieve a past decision (if any) and past-seen ballot ids. Tries to know of past decisions.
- **Leader fails**
  - Start another round
- **Messages dropped**
  - If too flaky, just start another round
- **Note that anyone can start a round any time**
- **Protocol may never end**
  - Impossibility result not violated
  - If things go well sometime in the future, consensus reached



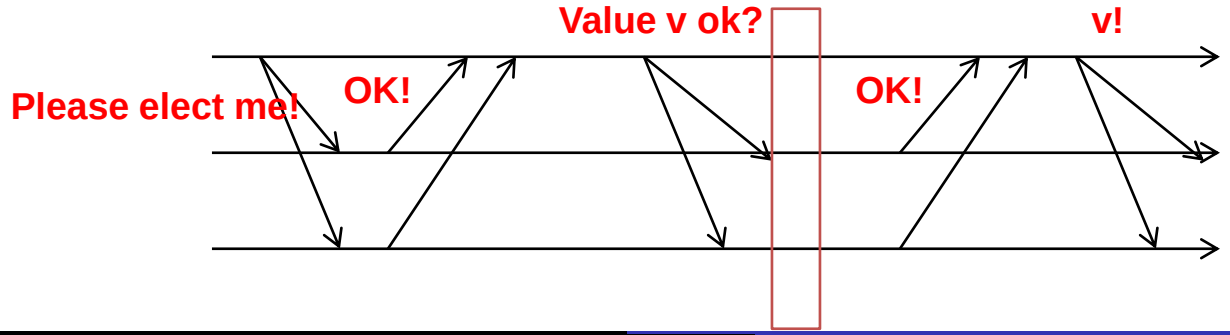
What could go Wrong?

- A lot more!
- This is a highly simplified view of Paxos.
- Lamport's original paper:

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems.



Conclusion

- Consensus is a very important problem
  - **Equivalent to many important distributed computing problems that have to do with reliability**
- **Consensus is possible to solve in a synchronous system where message delays and processing delays are bounded**
- **Consensus is impossible to solve in an asynchronous system where these delays are unbounded**
- **Paxos protocol:** widely used implementation of a safe, eventually-live consensus protocol for asynchronous systems
  - Paxos (or variants) used in Apache Zookeeper, Google's Chubby system, Active Disk Paxos, and many other cloud computing systems