CSE 486/586 Distributed Systems

Consensus

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Recap: Finger Table
• Finding a <key, value> using fingers

Let's Consider This...

One Reason: Impossibility of Consensus
• Q: Should Steve give an A to everybody taking CSE 486/586?
• Input: everyone says either yes/no.
• Output: an agreement of yes or no.
• Bad news
  – Asynchronous systems cannot guarantee that they will reach consensus even with one faulty process.
  – Many consensus problems
    – Reliable, totally-ordered multicast (what we saw already)
    – Mutual exclusion, leader election, etc. (what we will see)
    – Cannot reach consensus.

The Consensus Problem
• N processes
• Each process p has
  – input variable \( x_p \): initially either 0 or 1
  – output variable \( y_p \): initially b (b=undecided) – can be changed only once
• Consensus problem: Design a protocol so that either
  – all non-faulty processes set their output variables to 0
  – Or all non-faulty processes set their output variables to 1
  – There is at least one initial state that leads to each outcomes 1 and 2 above

Assumptions (System Model)
• Processes fail only by crash-stopping
• Synchronous system: bounds on
  – Message delays
  – Max time for each process step
    – e.g., multiprocessor (common clock across processors)
• Asynchronous system: no such bounds
  – E.g., the Internet

Assurance that a non-faulty process will see the input variable (b) set by another process is also unreasonable.

The Assumption of Asynchronous Systems

CSE 486/586

Consensus Question
• Q: Should Steve give an A to everybody taking CSE 486/586?
• Input: everyone says either yes/no.
• Output: an agreement of yes or no.
• Bad news
  – Asynchronous systems cannot guarantee that they will reach consensus even with one faulty process.
  – Many consensus problems
    – Reliable, totally-ordered multicast (what we saw already)
    – Mutual exclusion, leader election, etc. (what we will see)
    – Cannot reach consensus.

The Consensus Problem
• N processes
• Each process p has
  – input variable \( x_p \): initially either 0 or 1
  – output variable \( y_p \): initially b (b=undecided) – can be changed only once
• Consensus problem: Design a protocol so that either
  – all non-faulty processes set their output variables to 0
  – Or all non-faulty processes set their output variables to 1
  – There is at least one initial state that leads to each outcomes 1 and 2 above

Assumptions (System Model)
• Processes fail only by crash-stopping
• Synchronous system: bounds on
  – Message delays
  – Max time for each process step
    – e.g., multiprocessor (common clock across processors)
• Asynchronous system: no such bounds
  – E.g., the Internet

Assurance that a non-faulty process will see the input variable (b) set by another process is also unreasonable.

The Assumption of Asynchronous Systems

CSE 486/586

Consensus Question
• Q: Should Steve give an A to everybody taking CSE 486/586?
• Input: everyone says either yes/no.
• Output: an agreement of yes or no.
• Bad news
  – Asynchronous systems cannot guarantee that they will reach consensus even with one faulty process.
  – Many consensus problems
    – Reliable, totally-ordered multicast (what we saw already)
    – Mutual exclusion, leader election, etc. (what we will see)
    – Cannot reach consensus.

The Consensus Problem
• N processes
• Each process p has
  – input variable \( x_p \): initially either 0 or 1
  – output variable \( y_p \): initially b (b=undecided) – can be changed only once
• Consensus problem: Design a protocol so that either
  – all non-faulty processes set their output variables to 0
  – Or all non-faulty processes set their output variables to 1
  – There is at least one initial state that leads to each outcomes 1 and 2 above

Assumptions (System Model)
• Processes fail only by crash-stopping
• Synchronous system: bounds on
  – Message delays
  – Max time for each process step
    – e.g., multiprocessor (common clock across processors)
• Asynchronous system: no such bounds
  – E.g., the Internet

Assurance that a non-faulty process will see the input variable (b) set by another process is also unreasonable.

The Assumption of Asynchronous Systems

CSE 486/586

Consensus Question
• Q: Should Steve give an A to everybody taking CSE 486/586?
• Input: everyone says either yes/no.
• Output: an agreement of yes or no.
• Bad news
  – Asynchronous systems cannot guarantee that they will reach consensus even with one faulty process.
  – Many consensus problems
    – Reliable, totally-ordered multicast (what we saw already)
    – Mutual exclusion, leader election, etc. (what we will see)
    – Cannot reach consensus.

The Consensus Problem
• N processes
• Each process p has
  – input variable \( x_p \): initially either 0 or 1
  – output variable \( y_p \): initially b (b=undecided) – can be changed only once
• Consensus problem: Design a protocol so that either
  – all non-faulty processes set their output variables to 0
  – Or all non-faulty processes set their output variables to 1
  – There is at least one initial state that leads to each outcomes 1 and 2 above

Assumptions (System Model)
• Processes fail only by crash-stopping
• Synchronous system: bounds on
  – Message delays
  – Max time for each process step
    – e.g., multiprocessor (common clock across processors)
• Asynchronous system: no such bounds
  – E.g., the Internet

Assurance that a non-faulty process will see the input variable (b) set by another process is also unreasonable.

The Assumption of Asynchronous Systems

CSE 486/586
Example: State Machine Replication

• Run multiple copies of a state machine
• For what?
  – Reliability
• All copies agree on the order of execution.
• Many mission-critical systems operate like this.
  – Air traffic control systems, Warship control systems, etc.

First: Synchronous Systems

• For a system with at most \( f \) processes crashing, the
  algorithm proceeds in \( f+1 \) rounds (with timeout),
  using basic multicast (B-multicast).
• Values\(^i_r\): the set of proposed values known to
  process \( p_i \) at the beginning of round \( r \).
• Initially \( \text{Values}\(^0\) = \emptyset \); \( \text{Values}\(^1\) = \{v=x_p\} \)

for round \( r=1 \) to \( f+1 \) do
  multicast (Values\(^r\) )
  Values\(^r+1\) \leftarrow Values\(^r\) 
  for each \( V_j \) received
    Values\(^r+1\) = Values\(^r+1\) \cup V_j
  end
end

\( y_i=d_i = \min(\text{Values}\(^r+1\)) \)

Why Does It Work?

• Assume that two non-faulty processes differ in their
  final set of values \( \rightarrow \) proof by contradiction
• Suppose \( p_i \) and \( p_j \) are these processes.
• Assume that \( p_i \) possesses a value \( v \) that \( p_j \) does not possess.
• Intuition: \( p_j \) must have consistently missed \( v \) in all rounds. Let's backtrack this.
  – In the last round, some third process, \( p_k \), sent \( v \) to \( p_i \), and
    crashed before sending \( v \) to \( p_j \).
  – Any process sending \( v \) in the penultimate round must
    have crashed; otherwise, both \( p_k \) and \( p_j \) should have received \( v \).
  – Proceeding in this way, we infer at least one crash in
    each of the preceding rounds.
  – But we have assumed at most \( f \) crashes can occur and
    there are \( f+1 \) rounds \( \rightarrow \) contradiction.

Second: Asynchronous Systems

• Messages have arbitrary delay, processes arbitrarily slow
• Impossible to guarantee consensus even with a single process failure
  – Insight: a slow process is indistinguishable from a crashed process
• Impossibility applies to any protocol that claims to solve consensus
• 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

Are We Doomed?

• Asynchronous systems (i.e., systems with arbitrary delays) cannot guarantee that they will reach consensus even with one faulty process.
• Key word: “guarantee”
  – Does not mean that processes can never reach a consensus if one is faulty
  – Allows room for reaching agreement with some probability greater than zero
  – In practice many systems reach consensus.
• How to get around this?
  – Two key things in the result: faulty processes & arbitrary delays
Techniques to Overcome Impossibility

• Technique 1: masking faults (crash-stop)
  – For example, use persistent storage and keep local checkpoints.
  – Then upon a failure, restart the process and recover from the last checkpoint.
  – This masks fault, but may introduce arbitrary delays.

• Technique 2: using failure detectors
  – For example, if a process is slow, mark it as a failed process.
  – Enforce crash-stop: Actually kill it somehow, or discard all the messages from that point on (fail-silent)
  – This effectively turns an asynchronous system into a synchronous system
  – Failure detectors might not be 100% accurate and requires a long timeout value to be reasonably accurate.

Recall

• 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 (b=undecided)

• Consensus Problem: Design a protocol so that either
  – all non-faulty processes set their output variables to 0
  – Or non-faulty all processes set their output variables to 1
  – (No trivial solutions allowed)

Proof of Impossibility: Reminder

• State machine
  – Forget real time, everything is in steps & state transitions.
  – Equally applicable to a single process as well as distributed processes

  • A state (S1) is reachable from another state (S0) if there is a sequence of events from S0 to S1.
  • There an initial state with an initial set of input values.

Different Definition of “State”

• State of a process
• Configuration: = Global state. Collection of states, one per process; and state of the global buffer
• Each Event consists atomically of three sub-steps:
  – receipt of a message by a process (say p), and
  – processing of message, and
  – sending out of all necessary messages by p (into the global message buffer)

• Note: this event is different from the Lamport events
• Schedule: sequence of events
**State Valencies**

- Let config. C have a set of decision values V reachable from it
  - If |V| = 2, config. C is bivalent
  - If |V| = 1, config. C is said to be 0-valent or 1-valent, as is the case
- Bivalent means that the outcome is unpredictable (but still doesn’t mean that consensus is not guaranteed). Three possibilities:
  - Unanimous 0
  - Unanimous 1
  - 0’s and 1’s

---

**Guaranteeing Consensus**

- If we want to say that a protocol guarantees consensus (with one faulty process & arbitrary delays), we should be able to say the following:
  - Consider all possible input sets (i.e., all initial configurations).
  - For each input set (i.e., for each initial configuration), the protocol should produce either 0 or 1 even with one failure for all possible execution paths (runs).
    - i.e., no “0’s and 1’s”
- The impossibility result: We can’t do that.
  - i.e., there is always a run that will produce “0’s and 1’s”.

---

**Lemma 1**

Schedules are commutative

- Each schedule s can be applied to C
- Involve disjoint sets of receiving processes

---

**The Theorem**

- Lemma 2: There exists an initial configuration that is bivalent
- Lemma 3: Starting from a bivalent config., there is always another bivalent config. that is reachable
- Insight: It is not possible to distinguish a faulty node from a slow node.
- Theorem (Impossibility of Consensus): There is always a run of events in an asynchronous distributed system (given any algorithm) such that the group of processes never reaches consensus (i.e., always stays bivalent)

---

**Summary**

- Consensus: reaching an agreement
- Possible in synchronous systems
- Asynchronous systems cannot guarantee.
  - Asynchronous systems cannot guarantee that they will reach consensus even with one faulty process.
Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).