Consensus

CMPS 4760/6760: Distributed Systems
Overview

- Distributed Mutual Exclusion (15.2)
- Leader Election (15.3)
- Group communication (6.2, 15.4)
- Consensus (15.5, 21.5.2)
Consensus

- Problem definition (15.5.1)

- Consensus in synchronous systems
  - Consensus under crash failures (15.5.2)
  - Byzantine generals problem (15.5.3)

- Consensus in asynchronous systems
  - FLP Impossibility Result (15.5.4)

- Paxos (21.5.2)
Consensus Problem

- **Problem**: a collection of processes need to **agree** on a value after one or more of them has proposed what that value should be

- Reaching agreement is a fundamental requirement in distributed computing
  - Leader election / Mutual Exclusion
  - Commit or Abort in distributed transactions
  - Reaching agreement about which process has failed
  - Air traffic control system: all aircrafts must have the same view
Consensus

- Each process $p_i$ beings in undecided and proposes value $v_i \in D$.
- Processes exchange values with each other via message passing.
- $p_i$ enters the decided state by setting the value of a decision variable $d_i$ (write-once).
Requirements

- **Termination**: Eventually each correct process sets its decision variable

- **Agreement**: For any two processes $p_i$ and $p_j$, if they are correct and have entered the *decided* state, then $d_i = d_j$

- **Integrity**: If the correct processes all proposed the same value $v$, then for any correct process $p_i$ in the *decided* state, $d_i = v$
Assumptions

- $N$ processes, message passing only
- Communication is reliable
- Processes can fail: crash and byzantine
- Up to some number $f$ of $N$ processes are faulty
- Messages are not signed (‘oral’ messages)
When processes cannot fail

- A simple solution to solve consensus:
  - Each $p_i$ reliably multicasts its proposed value to the group
  - Each $p_i$ waits until it has collected all $N$ values and then sets $d_i = \text{majority}(v_1, v_2, \ldots, v_N)$
    - If no majority exists, $\text{majority}(v_1, v_2, \ldots, v_N) = \bot$
    - Other functions can also be applied, e.g., min or max for values that are ordered
When processes can fail

- Can we always achieve consensus if processes can crash?
  - Yes if the system is synchronous
  - No if the system is asynchronous even even with a single process failure (FLP impossibility result)

- What if process can fail in arbitrary (Byzantine) ways?
  - No if the system is asynchronous
  - Yes if the system is synchronous and $N > 3f$
Consensus and RTO-Multicast

- Implementing consensus using RTO-multicast
  - Each $p_i$ multicasts its proposed value to the group using RTO-multicast
  - Each $p_i$ sets $d_i =$ the first value it delivers.

- Implementing RTO-multicast using consensus [Chandra and Toueg 1996]
Byzantine Generals

Coordinated Attack Leading to Victory

Uncoordinated Attack Leading to Defeat
Byzantine Generals

- Three or more generals are to agree to attack or to retreat
- One, the commander, issues the order
- The others, lieutenants to the commander, decide whether to attack or retreat
- Both the commander and the generals can be treacherous
Byzantine Generals

- **Termination**: Eventually each correct process sets its decision variable
- **Agreement**: For any two processes $p_i$ and $p_j$, if they are correct and have entered the *decided* state, then $d_i = d_j$
- **Integrity**: If the commander is correct, then all the processes decide on the value that the commander proposed
Byzantine Generals and Consensus

- **BG from C**: We can construct a solution to BG from C as follows:
  - The commander $p_j$ sends its proposed value to itself and each of the lieutenants ($p_j$ may be faulty)
  - All generals run C with the values $v_1, v_2, ..., v_N$ that they receive

- **C from BG**: exercise
Consensus

- Problem definition (15.5.1)

- Consensus in synchronous systems
  - Consensus under crash failures (15.5.2)
  - Byzantine generals problem (15.5.3)

- Consensus in asynchronous systems
  - FLP Impossibility Result (15.5.4)

- Paxos (21.5.2)
Consensus in a Synchronous System with Crash Failures

- At most $f$ processes crash ($f$ is known)
- The algorithm operates in $f + 1$ rounds
- In each round, $P_i$ waits for a message from every $P_j$ for some fixed predetermined time
Consensus in a Synchronous System with Crash Failures

At most $f$ processes crash ($f$ is known)

The algorithm operates in $f + 1$ rounds

In each round, $P_i$ waits for a message from $P_j$ for some fixed predetermined time

\[
P_i::
\]

\[
\text{var}
\]

\[
V: \text{set of values initially } \{v_i\};
\]

\[
\text{for } k = 1 \text{ to } f + 1
\]

\[
\text{send } \{v \in V \mid P_i \text{ has not already sent } v\} \text{ to all;}
\]

\[
\text{receive } S_j \text{ from all processes } P_j, j \neq i;
\]

\[
V = V \cup S_j;
\]

\[
d_i = \min\{V\};
\]
Consensus in a Synchronous System with Crash Failures

- The simple algorithms guarantees
  - **Termination**: each correct process terminates in \( f + 1 \) rounds
  - **Integrity**: set \( V \) contains only the proposed values
  - **Agreement**: Let \( V_i \) denote the set of values of \( P_i \) after the round \( f + 1 \)

  Claim: If any value \( x \) is in the set \( V_i \) for some correct process \( P_i \), then it is also in
  the set of \( V_j \) of any other correct process \( P_j \)

  - Case 1: \( x \) was added to \( V_i \) in a round \( k < f + 1 \)
  - Case 2: \( x \) was added to \( V_i \) in the round \( f + 1 \)

- Message complexity: \( O((f + 1)N^2) \)
Byzantine Generals

Coordinated Attack Leading to Victory

Uncoordinated Attack Leading to Defeat
Impossibility with three processes

Impossibility with $N \leq 3f$: a reduction from the three-process case

- Consensus can be reached for $N \geq f + 2$ using signed messages
Solution with one faulty process

- $N \geq 4, f = 1$

Round 1: the commander sends a value to each of the lieutenants
  - The value can be different to different lieutenants if the commander is faulty

Round 2: each of the lieutenants sends the value it received to its peers
  - The value sent can be different from the value received if the lieutenant is faulty

Each lieutenant applies the majority function to the set of values it receives
Four Byzantine Generals

Faulty processes are shown coloured
Solution with $f$ faulty process

- The general algorithm operates over $f + 1$ rounds (Lamport et al. 1982) – best possible for deterministic solutions

- The general algorithm has a message complexity of $O(N^{f+1})$
  - can be improved using signed messages.
Consensus

- Problem definition (15.5.1)
- Consensus in synchronous systems
  - Consensus under crash failures (15.5.2)
  - Byzantine generals problem (15.5.3)
- Consensus in asynchronous systems
  - FLP Impossibility Result (15.5.4)
- Paxos (21.5.2)
FLP Impossibility Result

- One of the most important results in distributed computing

Michael J. Fischer  Nancy A Lynch  Michael S. Paterson
Models and Assumptions

- Consider an easier problem and a more restrictive system model.
- Each process $i$ proposes a binary value $v_i$ in $\{0, 1\}$.
- A weaker termination requirement: some process eventually enters the decision state.
- A weaker integrity requirement: both values 0 and 1 should be possible outcomes.
- Only one process crashes (and we can choose which one).
Network

- A global message buffer

- If `receive()` is performed an unbounded number of times, then every message is eventually delivered
States

- Global state $G$: state of all the processes and the state of the global buffer
- $G[i]$: state of process $i$, initially this includes the proposed value $v_i$ and the decision variable $d_i = \bot$ (undecided)

- An event $(p, m)$ consists of
  - receipt of a message $m$ by a process $p$
  - processing of $m$ (may change recipient’s state)
  - sending out of all necessary messages by $p$

- Schedule: sequence of events
event $e_1 = (p, m)$

event $e_2 = (p', m')$

schedule $s = e_1 e_2$
Commute property of disjoint events

\[ st(G) = ts(G) \] if \( s \) and \( t \) involve disjoint set of receiving processes and are each applicable to \( G \)
Asynchrony of events

- Any event may be arbitrarily delayed

If $e$ is enabled at $G$ and $e \notin s$, it is still enabled at $s(G)$
Main Idea of the Proof

- There is an initial global state in which the system is **indecisive**
- There exists a method to keep the system **indecisive**
- How to model **indecision**?
Bivalent states

- $G.V$ — set of decision variables of global state reachable from $G$
  - If $G.V = \{0\}$, $G$ is 0-valent
  - If $G.V = \{1\}$, $G$ is 1-valent
  - If $|G.V| = 2$, $G$ is bivalent (indecisive)

- Lemma 1: Every consensus protocol has a bivalent initial global state
Suppose all initial global states were either 0-valent or 1-valent.

The protocol must have both 0-valent or 1-valent states (from integrity)

Place all initial global states side-by-side (in a lattice), where adjacent states differ in proposed values for exactly one process

Claim: there has to be some adjacent pair of 1-valent and 0-valent global states.
Poof of Lemma 1 (cont.)

- There has to be some adjacent pair of 1-valent and 0-valent global states.
- Assume that they differ in the state of $p$ and let $p$ be the process that has crashed (i.e., is silent throughout)
- Both initial states will reach the same decision value for the same sequence of events, a contradiction
Main Idea of the Proof

- Lemma 1: Every consensus protocol has a bivalent initial global state

- Lemma 2: Starting from a bivalent global state, there is always another bivalent global state that is reachable
Proof of Lemma 2

- Let $G$ be a bivalent global state of a protocol.
- Let $e = (p, m)$ be an event applicable to $G$.
- Let $C$ be the set of global states reachable from $G$ without applying $e$.
- Let $D = e(C)$.
  - Applying asynchrony of events
- Claim: $D$ contains a bivalent global state.
- Proof by contradiction. Assume $D$ does not contain a bivalent global state.
Proof of Lemma 2 (cont.)

- Claim 1: $D$ contains both 0-valent and 1-valent states
- Claim 2: There are two states $C_0$ and $C_1$ in $C$ such that
  
  - $C_1 = f(C_0)$ for some event $f$
  - $D_0 = e(C_0)$ is 0-valent, $D_1 = e(C_1)$ is 1-valent

Proof of Claim 2: consider the shortest sequence $t$ without applying $e$ such that $et(G)$ has different valency from $e(G)$. Let $C_0$ and $C_1$ be the last two states reached in this sequence.
Proof of Lemma 2 (cont.)

- Let $C_1 = f(C_0)$ and $f$ be an event on process $q$
- Case 1: $p \neq q$

$f$ is applicable to $D_0$ by commutativity of disjoint events
=> a contradiction since $D_0$ is 0-valent and $D_1$ is 1-valent
Proof of Lemma 2 (cont.)

- Let $C_1 = f(C_0)$ and $f$ be an event on process $q$
- Case 2: $p = q$

$s$: a finite deciding run from $C_0$ in which $p$ takes no steps

$K$ is bivalent, contradicting the fact that $s$ is a deciding run
Putting it all together

- **Lemma 1:** Every consensus protocol has a bivalent initial global state

- **Lemma 2:** Starting from a bivalent global state, there is always another bivalent global state that is reachable

- **Theorem (Impossibility of Consensus):** There is always a run of events in an asynchronous distributed system such that the group of processes never reach consensus (i.e., stays bivalent all the time)
Consensus

- Problem definition (15.5.1)
- Consensus in synchronous systems
  - Consensus under crash failures (15.5.2)
  - Byzantine generals problem (15.5.3)
- Consensus in asynchronous systems
  - FLP Impossibility Result (15.5.4)
- Paxos (21.5.2)
Paxos Algorithm

- Invented by Leslie Lamport
- A family of protocols providing distributed consensus
- Most popular “consensus-solving” algorithm
- Safety is provided: agreement and integrity
- Liveness is not: no guarantee on termination
  - Provides eventual liveness if majority of of the processes run for long enough with sufficient network stability
- A lot of systems use it
  - Zookeeper (Yahoo!), Google Chubby, and many other companies
Paxos Algorithm in Chubby (21.5.2)

- Operate over a set of replicas managed by different servers
- Any replica can submit a value with the goal of achieving consensus on a final value
- Agreement equates to all replicas having this value as the next entry in their update logs (achieving a consistent view of the logs)
Message exchanges in Paxos (in absence of failures) - step 1

Step 1: electing a coordinator

- Propose (seq_number)
- Promise

Coordinator → Replicas
Message exchanges in Paxos (in absence of failures) - step 2

Step 2: seeking consensus

Coordinator

Accept (value)

Acknowledgement

Replicas
Message exchanges in Paxos (in absence of failures) - step 3

Step 3: achieving consensus

Coordinator  Commit  Replicas