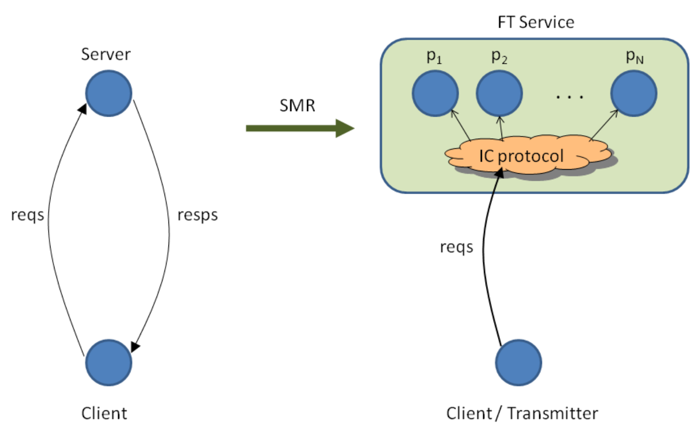
# CS 5410: Distributed Systems
# Lectures: Asynchronous Consensus

Chi Ho

October 28, 2009
November 2, 2009

## 1 State Machine Replication Approach

Figure 1: State machine replication (SMR) approach.




- Client-service systems, where services are deterministic, can be turned fault tolerant by the state machine replication approach (Figure 1).
- Each non-faulty server processes all requests in the same order:
  - **Agreement:** Every non-faulty server receives the same set of requests.
  - **Order:** Every non-faulty server processes the requests in the same relative order.
- Rephrase the system as a set of processes, one designated as the *transmitter*.
- Agreement requirements: (Interactive Consistency / Byzantine Agreement)

- **IC1:** All non-faulty processes agree on the same value.
- **IC2 (non-triviality):** If the transmitter is non-faulty, then all non-faulty processes use the transmitter's value as the one they agree on.

- We've seen solutions:
  - Fail-stop processes. (implicitly synchronous.)
  - Byzantine processes:
    * Synchronous clocks + bounded delivery delays.
    * Message authentication.

## 2 Today Outline

- Remove synchrony ⇒ IC is unsolvable!

## 3 Consensus problem

A consensus protocol involves a group of processes, where each process has an *initial value* in {0, 1}. The protocol coordinates the group of processes to produce a *decision*, which is also in {0, 1}.

- *Agreement.* All non-faulty processes that make a decision must choose the same value.
- *Validity.* If all processes have the same initial value, then the decision will be the initial value.
- *Integrity.* Every correct process decides at most once, and its decision must be proposed by some process.
- *Termination.* Every non-faulty process must eventually decide on a value.

**Consensus unsolvable ⇒ IC unsolvable:**

- (Contrapositive) Solve IC ⇒ solve consensus:
  - Each $p_i$ uses FTB to broadcast $v_i$.
  - Each $p_i$ uses a deterministic function to select a value from the set of values it received.

## 4 Asynchronous Environment

Asynchronous environment:

- No assumptions about the relative speeds of processes.
- No assumptions about the delay time in delivering a message.

Implications:

- No ability to detect the failure of a process. (i.e., fail appears the same as very slow.)
- If wait for a process and it crashes, may wait forever.
- If not wait for a process, the process may have already decided one value and the rest of the processes will go on and decide a different value.

## 5 FLP's Impossibility Result

FLP: Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson.

Given a system that works with:

- asynchronous environment,
- at most one process failure,
- protocol steps are deterministic.

Emphasizing that we give the strongest assumptions possible:

- Failure model: crash failures only,
- Communication model: reliable channels,
- Connectivity model: full connectivity.

### 5.1 Definitions

**Definition 1 (Indistinguishability)** *Two global states are indistinguishable to a process if they appear the same to the process.*

**Definition 2 (*Valency* of a global state)** *A global state is called:*

- *0-valent: decision will be 0.*
- *1-valent: decision will be 1.*
- *uni-valent: either 0-valent or 1-valent.*
- *bi-valent: decision can be either 0 or 1.*

### 5.2 Initial Bi-valency

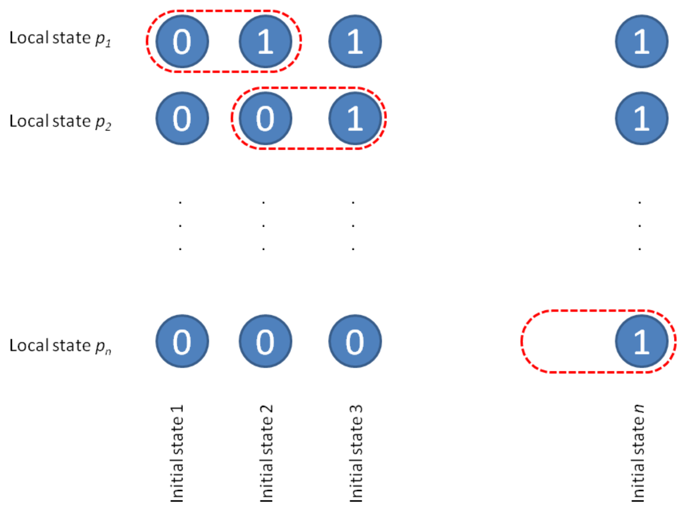
Lemma 1 *There exists a bi-valent initial global state.*

Proof: Show by contradiction using the example in Figure 2:

- Leftmost initial state must decide 0 by Validity.
- Second leftmost global state must decide 0 because it is indistinguishable from leftmost one.
- By induction, rightmost global state must decide 0. This is a contradiction to Validity (all processes has 1, hence must decide 1.)

■

Figure 2: Indistinguishable Initial Global States. Two adjacent states are indistinguishable when the process in dashed oval is slow or has crashed.




### 5.3 Procrastination

Notations:

- $m(\Sigma)$ — the state resulting from $m$ being received in state $\Sigma$.
- $s(\Sigma)$ — the state resulting from messages in $s$ begin received in state $\Sigma$ in the sequential order.
- $s_1 : s_2$ — sequence $s_1$ concatenated with sequence $s_2$.

Lemma 2 *Given any global state $\Sigma$, any message $m$:*

- *$\Sigma$ is bi-valent,*
- *$m(\Sigma)$ is uni-valent,*

*there exists a sequence $s^*$ of messages such that $m(s^*(\Sigma))$ is bi-valent.*

Proof:

$$S = \{\text{finite sequences of messages (w/o } m\text{) that can be received in } \Sigma\}$$

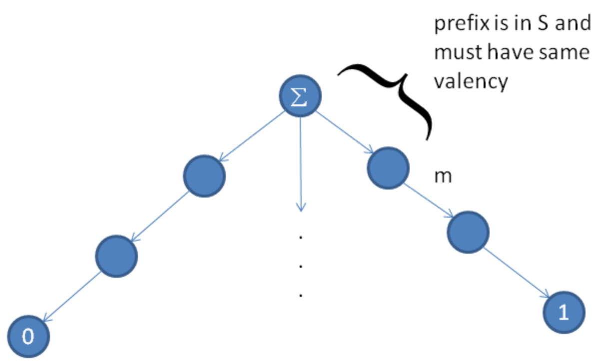
$$S \neq \emptyset \quad \text{(at least contains the empty sequence)}$$

- Term: valency of a sequence $s$ = valency of $m(s(\Sigma))$ .
- If exists $s^* \in S$ such that is bi-valent, we're done.
- Show: Otherwise, there will be a contradiction.
  - $S$ contains both 0-valent and 1-valent sequences. (Explained in Figure 3.)

- $\exists s, s' \in S : (s = s' : m') \wedge (s$ is 0-valent$) \wedge (s'$ is 1-valent$)$. (Figure 4 shows how to find $s$ and $s'$.)
- There are two cases: $m'$ is received by $p$ (receiver of $m$) and not.
  - Case 1 ($m'$ received by $p$): Consider a sequence $a$ after $s'$ that (1) does not contain messages received by $p$, and (2) $a : s'$ is uni-valent. This sequence must exist because otherwise the system would stuck when $p$ fails. Without loss of generality, assume that the decision is 0. Then other processes decide 0, while $p$ can receive $m'$ then $m$ and decide 1. Contradicting to Agreement.
  - Case 2: Let $p'$ be the receiver of $m'$. From the same state ($s'(\Sigma)$), the relative ordering of $p'$'s receiving $m'$ and $p$'s receiving $m$ should lead to the same decision, because they proceed independently, and because of the assumption that steps are deterministic. In this case, different decisions comes with different relative ordering. This is a contradiction.
- Since all cases lead to contradictions, the assumption can't be true. We can conclude now that $S$ contains a bi-valent sequence. Let $s^*$ be the bi-valent sequence.

■

Figure 3: Because $\Sigma$ is bi-valent, there are sequences that lead to both 0-valent and 1-valent states. If such a sequence does not contains $m$, it is in $S$. If it does contain $m$, then its prefix before $m$ is in $S$ and has the same valency.




## 5.4 Impossibility

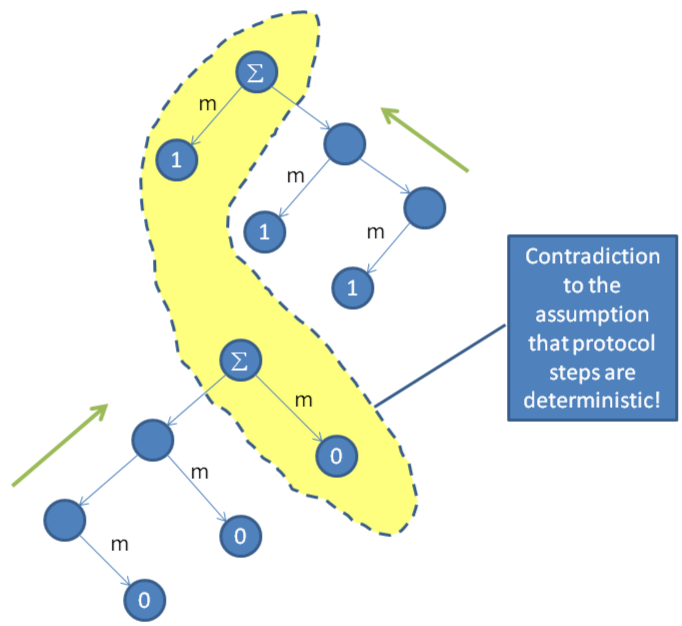
Theorem 1 (FLP's Result) *No consensus protocol is totally correct in spite of one fault.*

Proof:

- Lemma 1 ⇒ exists an bi-valent initial global state.
- Lemma 2 ⇒ any time a global state is about to change from bi-valent to uni-valent asynchrony may keep it bi-valent.
- Hence the protocol may never terminate.

■

Figure 4: Find $s$ and $s'$ by going back two sequences of different valencies in $S$. The worst case is when we go back all the way on both branches (sequences) without finding $s$ and $s'$. But this is not possible.




—

## 6 Getting Around FLP

To get around FLP we need to understand what lead to FLP and change them:

- Agreement:
  - $\epsilon$-consensus:
    - Real input values,
    - Values agreed upon are within $\epsilon$ of each other.
  - k-set consensus: no more than $k$ decisions.
- Termination:
  - Terminate with arbitrary large probability.
  - Change from "all runs terminate" to "always exist a run that terminate".
- Asynchrony:
  - Strengthen the timing model: partial synchrony; Termination during period of synchrony. (Break Procrastination.)
  - Strengthen the system model: add *failure detectors* to distinguish between processes that crash and that slow. (Break Indistinguishability.)
- Existing of a bi-valent initial state: control the input values so that initial state is uni-valent.

# 7 Getting Around #1: Randomization

Randomization assumption:

- Network delivery is random: if there are $n$ messages sent to a process $p$, then $p$ can deliver the messages in any of the $(n!)$ orders with a non-zero probability.

Change Termination to:

- *Termination.* Every non-faulty process must decide on a value with probability 1.

---

**Algorithm 1** One third consensus protocol. $n = 3t + 1$.

---

$r_i = 0$;
$M_i = \emptyset$;
**while** (not decided) **do**
  $r_i + +$;
  **if** ($M_i$ has one value) **then**
    decide $M_i$;
  **end if**
  broadcast $(r_i, v_i)$;
  $M_i$ = wait for messages $(r_i, *)$ from $n - t$;
  $v_i = majority(M_i)$;

**end while**

---

Correctness:

- Agreement: Consider any two processes $p_i$ and $p_j$ that decide. There are two cases: $p_i$ and $p_j$ decide in the same round, and not.

  1. Case 1: $p_i$ and $p_j$ decide in round $r$.
     - $p_i$ decides $v$ in round $r \Rightarrow$ there are $n-t$ processes broadcasting $(r-1, v)$ in previous round. Let $V$ be the set of such processes.
     - $p_j$ decides $v'$ in round $r \Rightarrow$ there are $n-t$ processes broadcasting $(r-1, v')$ in previous round. Let $V'$ be the set of such processes.
     - $(|V| = n - t) \wedge (|V'| = n - t) \wedge (n = 3t + 1) \Rightarrow |V \cap V'| > 1$
     - A process in $V \cap V'$ only broadcast a message in round $r - 1 \Rightarrow v = v'$.
  2. Case 2: $p_i$ decides $v$ in round $r$, $p_j$ decides $v'$ in round $r' > r$.
     - $p_i$ decides $v$ in round $r \Rightarrow$ there are $n - t$ processes broadcasting $(r - 1, v)$ in round $r - 1$.

     $\Rightarrow$ Each process $p_k$ receives at least $n-2t$ messages of the form $(r-1, v)$ in round $r-1$.

     $\Rightarrow$ $p_k$ sets $v_k$ to $v$ in round $r-1$ (because $n-2t = t+1$ messages $(r-1, v)$ is a majority among $n - t = 2t + 1$ messages).

     $\Rightarrow$ $p_k$ broadcasts $(r, v)$ in round $r$.

⇒ Every process, including $p_j$, receives $n - t$ messages $(r, v)$ in round $r$ and decides $v$ in round $r + 1$.

⇒ $v' = v$ as desired.

- Validity: every process has the same initial values $v \Rightarrow$ every $p_i$ has $M_i = \{(1, v)\}$ in round 1 and decides $v$ in round 2.
- Integrity:
  - Every process that decides will not go on to the next iteration of the main loop ⇒ decide at most once.
  - Every value in any $M_i$ is from a process ⇒ decision is among the initial values.
- Termination: One case that leads to termination is that every process $p_i$ has the same $M_i$ in the same round. This can occur when every process $p_i$ delivers the same sequence of messages in a round. We will show that this occurs with probability 1 when the number of rounds approaches infinity.
  - In 1 round, there are $n$ messages $m_1, m_2, \ldots, m_n$ broadcast. By the network-randomization assumption, any delivery order, say $m_{r_1} m_{r_2} ... m_{r_n}$, will have a non-zero probability $\rho$.
  - The processes deliver the messages in the same order $m_{r_1} m_{r_2} ... m_{r_n}$ with probability $\rho^n$.
  - The complement (the messages are not delivered in the order $m_{r_1} m_{r_2} ... m_{r_n}$ by all processes) is therefore $(1 - \rho^n)$.
  - The probability of all processes not delivering the same sequence of messages in $k$ rounds is $(1 - \rho^n)^k$.
  - Since $(1-\rho^n)^k \to 0$ when $k \to \infty$, all processes will deliver the same sequence of messages in a round with probability 1, as desired.

Characteristics:

- Not optimal in resilience: $n = 3t + 1$;
- Very simple.