Distributed algorithms: lesson 3 on the consensus impossibility result

Emmanuelle Anceaume

http://people.irisa.fr/Emmanuelle.Anceaume/
Content of this lesson

- We will focus on
  - The impossibility to solve Consensus in asynchronous environments
Impossibility of asynchronous agreement

- Easy argument that says that you cannot do most things in an asynchronous system with $n/2$ crash failures
  - Partition the system in two sets of processes
  - One cannot tell the difference between slow and crashed processes
- The Fisher-Lynch-Paterson (FLP) result says something stronger
  - You cannot do agreement in an asynchronous message-passing system if even 1 crash is allowed
- ... unless you add randomization
Theorem (The « FLP » impossibility result)

*There exists no deterministic algorithm that solves the binary consensus problem in the presence of even if a single faulty process*.

---

No deterministic protocol can solve Consensus in finite time if at most an entity can crash

- The proof of this result is neither simple nor complex.
- It uses some constructs that will be very useful in other situations also.
- We will need to define precisely the entire environment, including executions, events, configurations, ...
Each process $p_i$ has a one-bit input register $x_{p_i}$, and a write-once output register $y_{p_i}$ with values in \{0, 1, b\}, and unlimited internal storage.

Each process is modeled as an automata.

The state of process $p_i$ comprises the value of $x_{p_i}$, the value of $y_{p_i}$, all the messages it has received, and all its internal storage.

- Initially at $p_i$: $x_{p_i} = 0$ or $x_{p_i} = 1$ and $y_{p_i} = b$
- Decision states: $y_{p_i} = 0$ or $y_{p_i} = 1$

The goal is to have all non-faulty processes $p_i$ set, in finite time, their output registers to the same value in \{0, 1\} subject to the nontriviality condition (i.e., if all input values are the same, then the value of $y_{p_i}$ must be that value).
Processes communicate by exchanging messages

- Processes are modeled as automata that communicate by means of messages.
- A message is a pair \((p, m)\) where \(p\) is the recipient of \(m\) and \(m\) is some message value.
- The communication system maintains a buffer of messages that have been sent but not yet delivered.
- Two communication operations are provided:
  - **Send**\((p, m)\) : places \((p, m)\) in the message buffer.
  - **Receive**\((p)\) :
    - Deletes some message \((p, m)\) from the buffer and returns \(m\) to \(p\). We say that \((p, m)\) is delivered.
    - Or returns null and leave the buffer unchanged, even if the buffer contains some message \((p, m)\).
Processes communicate by exchanging messages

- The message system acts in a non-deterministic way but is reliable (does not lose messages)
- $\text{Future}(t) = \text{All the messages which have been sent but not yet delivered by time } t$

```
8 / 76
```
Processes communicate by exchanging messages

- **Fairness property**
  - If queried infinitely many times by non-faulty process $p$, every message $(p, m)$ is eventually delivered.

- Since messages are not lost, once $(p, m)$ is enabled in some configuration, it is enabled in all successor configurations until $m \neq \emptyset$.

- So to ensure fairness, we need to ensure that $p$ eventually performs $(p, m)$ and $m \neq \emptyset$. 

A configuration $C$ (or global state) of the system at time $t$ is a snapshot of the system at time $t$:
- it contains the state $s_i$ of each process $p_i$ and the set $\text{Future}(t)$ of the future events that have been generated so far
- $C = (s, \text{Future}(t))$ with $s = (s_1, s_2, \ldots, s_n)$

An initial configuration is a configuration in which each process starts at an initial state and the message buffer is empty
• An **event** $e$
  - Brings the system from a configuration $C$ to another configuration $C'$, and
  - Involves a single process $p_i$

• An event can be
  - Either the receipt of a message $e = (p_i, m)$
  - Or the crash of $p_i$ denoted by $e = (p_i, \text{crash})$

• Given an event $e$ and a configuration $C$ then
  - $C' = e(C)$ is the configuration obtained when applying event $e$ to configuration $C$
Crash event

- To describe the fact that some process may fail by crashing during an execution, we have
  \[ Future(0) = \{(p_1, \text{crash}), (p_2, \text{crash}), \ldots, (p_n, \text{crash})\} \]

- We are interested in execution in which at most a single crash can occur
- If event \((p_i, \text{crash})\) occurs at time \(t\) then
  - Process \(p_i\) stops running
  - All the other crash events are removed from \(Future(t)\)
  - All the message \((p_i, -)\) are removed from \(Future(t')\) with \(t' \geq t\)
If the receipt event \( e = (p_i, m) \) is generated then process \( p_i \) executes the following atomic sequence of actions

- First \( p_i \) executes \( \text{receive}_i(p_i) \); recall that \( \text{receive}_i(p_i) \) returns \( m \in \{\text{Future}, \bot\} \)
  - if \( m \neq \bot \), then \( (p_i, m) \) is removed from \( \text{Future} \)
- Based on its local state and \( m \), \( p_i \) enters a new state
- \( p \) may send some messages \( m' \) to some processes \( q \), i.e., puts \( (q, m') \) to \( \text{Future} \)
A **schedule** is a finite or infinite linear sequence of events $e_1 e_2 \ldots e_k$ taken by the processes from a given configuration of the system.
Decision configuration

- If a process $p_i$ sets its output register $y_{p_i}$ to 0 or 1, we say that $p_i$ has decided, and its state is a decision state.
- A configuration in which some non-faulty processes have decided on the same value is called a decision configuration.
- Depending on its value, a decision configuration is a 0-decision or 1-decision.
- Once a process makes a decision it cannot change it anymore.
Let $C$ be any configuration and $C(C)$ be the set of all configurations reachable from $C$.

If all decision configurations in $C(C)$ are 0-decision (resp. 1-decision), we say that $C$ is 0-valent (resp. 1-valent).

Thus in a $v$-valent configuration, whatever happens, the decision is going to be on $v$.

If, instead, there are both 0-decision and 1-decision configurations in $C(C)$, then we say that $C$ is bivalent.

Thus in a bivalent configuration, which value is going to be decided depends on future events.
Proving the correctness of a distributed algorithm is a game

When designing fault-tolerant algorithms, we often assume the presence of an adversary that plays against the algorithm

- It has some control on the behavior of the system
- It knows the content of all sent messages
- It knows the local state of each process
- It is the scheduler:
  - It will select the next process to take a step
  - It will select the message the process will receive
  - It will select the process that will receive the crash event

Thus it controls the set *Future*

However

- It cannot prevent a message from being eventually received
- It cannot make more than one process crashed
The core of the FLP argument is a strategy allowing the adversary (who controls the scheduling) to steer the execution away from any univalent configuration.
Suppose that from some configuration $C$, the schedules $\sigma_1$ and $\sigma_2$ lead to configurations $C_1$, $C_2$, respectively.

- If the set of processes affected by events in $\sigma_1$ are disjoint from the set of processes affected by events in $\sigma_2$ then

- Both $\sigma_1 \sigma_2$ and $\sigma_2 \sigma_1$ can be applied to $C_1$, and $\sigma_1(\sigma_2(C)) = \sigma_2(\sigma_1(C))$
Commutativity property of schedules

Diamond pattern

Suppose that from some configuration $C$, the schedules $\sigma_1$ and $\sigma_2$ lead to configurations $C_1$, $C_2$, respectively.

- If the set of processes affected by events in $\sigma_1$ are disjoint from the set of processes affected by events in $\sigma_2$ then
- Both $\sigma_1 \sigma_2$ and $\sigma_2 \sigma_1$ can be applied to $C_1$, and $\sigma_1(\sigma_2(C)) = \sigma_2(\sigma_1(C))$
Fisher, Lynch and Paterson’s impossibility result

**Theorem**

No deterministic consensus protocol is correct in an asynchronous environment in spite of at most of crash.
Fisher, Lynch and Paterson’s impossibility result

Theorem

No deterministic consensus protocol is correct in an asynchronous environment in spite of at most one crash

Proof: The proof proceeds by contradiction. We assume that there is a protocol $\mathcal{P}$ that correctly solves the consensus problem, i.e., if we consider all possible executions of $\mathcal{P}$ in a complete communication graph with at most one crash, then within finite time all nonfailed processors decide
Proof of the theorem

The basic idea of the proof is to show circumstances under which the protocol remains forever indecisive.

1. We show that among the initial configurations, there is at least one in which the decision is not already pre-determined (i.e., bivalent configuration).

2. We then construct an execution that avoids ever taking a step that would engage the system to a particular decision.
Lemma (1)

Any consensus protocol that tolerates at least one faulty process has at least one bivalent initial configuration.

Proof: By contradiction. Suppose that all the initial configurations are univalent (i.e. are completely determined by the set of initial values)
Proof of the existence of bivalent initial configurations

Two initial configurations are adjacent if they differ by the initial value of a single process.

- For any two configurations $C$ and $C'$ (we have $n = 2^N$ possible initial configurations), it is always possible to find a chain of initial configurations, each adjacent to the next, starting with $C$ and ending with $C'$.
Proof of the existence of bivalent initial configurations

By the validity property,

- The initial configuration where all the initial values are 0 is 0-valent since by the validity property 0 must be decided
Proof of the existence of bivalent initial configurations

By the validity property,

- The initial configuration where all the initial values are 1 is 1-valent since by the validity property 1 must be decided.
Proof of the existence of bivalent initial configurations

- In this chain, there exists a 0-valent initial configuration $C_i$ adjacent to a 1-valent configuration $C_{i+1}$
- Let $p_i$ be the process whose initial value differ in $C_i$ and $C_{i+1}$
Proof of the existence of bivalent initial configurations

Let $C_0 = C_i$ and $C_1 = C_{i+1}$

Consider some deciding execution applicable to configuration $C_0$ in which process $p$ fails before sending any message

Let $\sigma$ be the associated sequence of events

- The first event of $\sigma$ is $(\text{crash}, p_i)$
Proof of the existence of bivalent initial configurations

Both runs are identical except for the initial value of $p_i$. Thus all the remaining processes must behave the same way and thus the decision state must be 0. This is a contradiction since the execution is 1-valent.

Since $\sigma$ is 0-valent the decision state is 0.

---

Diagram:

- States: $[0,0,\ldots,0]$, $[1,0,\ldots,0]$, $[1,1,\ldots,0]$, $[1,\ldots,1,0,0,\ldots,0]$, $[1,\ldots,1,1,0,\ldots,0]$, $[1,1,\ldots,1]$
- Transitions: $\sigma$
- Decision configuration

---
Proof of the existence of bivalent initial configurations

Since \( \sigma \) is 0-valent the decision state is 0.
Proof of the existence of bivalent initial configurations

- As $\sigma$ is a deciding sequence of events, eventually the same decision configuration is reached starting from $C_1$
- If it is a 1-decision configuration, then $C_0$ is bivalent
- Otherwise $C_1$ is bivalent
- In either cases, the assumed nonexistence of a bivalent initial configuration is contradicted

- The outcome of the consensus algorithm $\mathcal{P}$ is not uniquely predetermined by the initial configurations

Initial bivalent configuration

Any asynchronous consensus protocol that tolerates at least one faulty process has at least one bivalent initial configuration
We will show that if we delay a pending receipt event \( e = (p, m) \) in the message set for an arbitrarily amount of events, then there will be one configuration in which you deliver \( e \) and end up in a bivalent configuration.

By this we can see that guaranteeing that there is a path from one bivalent configuration to another one by delaying a message long enough raises the possibility that an infinite loop will develop and the protocol will remain forever undecided.
Lemma (3)

Let $C$ be a nonfaulty bivalent configuration of the protocol, and let $e = (p, m)$ be a receipt event that is applicable to $C$.

Let $C$ be the set of nonfaulty configurations reachable from $C$ without applying $e$.

Let $D$ be the set of configurations of the form $C \cdot e$ where $C \in C$.

Then $D$ contains a bivalent configuration.

- Observe that since event $e$ is applicable to $C$
- By definition of $C$ and because of the unpredictability of communication delays
  $\rightarrow$ $e$ is applicable to every configuration $C_i \in C$
Let $C$ be all possible configs reachable from $C$ without applying $e$. Apply event $e$ to all configs in $C$ and call the resulting configs $D$. 

![Diagram showing the relationship between $C$ and $D$]
Let $C$ be all possible configs reachable from $C$ without applying $e$  

Apply event $e$ to all configs in $C$ and call the resulting configs $\mathcal{D}$

Starting from a bivalent configuration

There is always a bivalent configuration that is reachable
Lemma 3

The proof is by contradiction

- We assume that every configuration $D \in \mathcal{D}$ is univalent
- Part 1: We will show that $\mathcal{D}$ contains both 0-valent and 1-valent configurations
- Part 2: We will show that $\mathcal{C}$ contains two neighbor configurations $C_0$ and $C_1$ that lead to respectively a 0-valent and 1-valent configuration in $\mathcal{D}$
- Part 3: Based on these two neighbor configurations we will reach a contradiction
Lemma 3

The proof is by contradiction

- We assume that every configuration $D \in \mathcal{D}$ is univalent
- Part 1: We will show that $\mathcal{D}$ contains both 0-valent and 1-valent configurations
- Part 2: We will show that $\mathcal{C}$ contains two neighbor configurations $C_0$ and $C_1$ that lead to respectively a 0-valent and 1-valent configuration in $\mathcal{D}$
- Part 3: Based on these two neighbor configurations we will reach a contradiction
$D$ contains both 0-valent and 1-valent configurations

By assumption of the Lemma, $C$ is a bivalent configuration
$D$ contains both 0-valent and 1-valent configurations

There must exist a 0-valent configuration $E_0$ reachable from $C$
$D$ contains both 0-valent and 1-valent configurations.

There must exist a 1-valent configuration $E_1$ reachable from $C$. 
Case 1: $E_i$, $i = 0, 1$, belongs to $C$ (that is event $e$ is not applied along $\sigma_i$) then $e$ can be applied to $E_i$.
$\mathcal{D}$ contains both 0-valent and 1-valent configurations

Let $D_i = E_i \cdot e$

- $D_0$ comes after $E_0$ which is 0-valent and since $\mathcal{D}$ contains only univalent configurations (by assumption) and $D \in \mathcal{D}$ then $D_0$ can only be 0-valent
- By the same argument applied to $E_1$, $D_1$ is 1-valent
\( D \) contains both 0-valent and 1-valent configurations

case 2 : \( E_i \) does not belong to \( C \) (that is event \( e \) has been applied along \( \sigma_i \), with \( i = 0, 1 \)).
\( \mathcal{D} \) contains both 0-valent and 1-valent configurations.

Thus there is a configuration \( C_i \in \mathcal{C} \) such that event \( e \) is applied to \( C_i \) and \( D_i = C_i \cdot e \), with \( D_i \in \mathcal{D} \).
\( \mathcal{D} \) contains both 0-valent and 1-valent configurations

- \( \mathcal{D} \) contains only univalent configurations (by assumption)
- \( D_0 \) leads to \( E_0 \) which is 0-valent, thus \( D_0 \) is 0-valent
- Same argument for \( D_1 \)
So far, we have shown that $\mathcal{D}$ must contain both 0-valent and 1-valent configurations if it contains no bivalent configurations.

We use this fact for the rest of the proof.
Lemma 3

The proof is by contradiction

- We assume that every configuration $D \in \mathcal{D}$ is univalent
- Part 1: We have shown that $D \in \mathcal{D}$ contains both 0-valent and 1-valent configurations
- Part 2: We will show that $\mathcal{C}$ contains two neighbor configurations $C_0$ and $C_1$ that lead to respectively a 0-valent and 1-valent configuration in $\mathcal{D}$
- Part 3: Based on these two neighbor configurations we will reach a contradiction
$\mathcal{C}$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $\mathcal{D}$

**Definition (Neighbor configurations)**

Two configurations $C_0$ and $C_1$ are neighbors if one results from the other by application of a single receipt event.
\( C \) contains two neighbor configurations \( C_0 \) and \( C_1 \) that lead to 0-valent \( D_0 \) and 1-valent \( D_1 \) in \( D \)

What do we want to prove?

\[
e = (p, m)
\]

\[
e' = (p', m')
\]

0-valent \( D_0 \)

1-valent \( D_1 \)
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $\mathcal{D}$

Let $C$ be the bivalent configuration of the lemma, and $C_0$ reachable from $C$ that leads to $D_0$ a 0-valent configuration of $\mathcal{D}$ by applying step $e$
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $D$

Since event $e$ is applicable from $C$ then one can apply this event all along the path from $C$ to $C_0$
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $D$.

All these configurations belong to $D$. Hence they are all univalent. Some of them can be 0-valent as is $D_0$.
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $\mathcal{D}$.

If one of them is 1-valent, we are done. We have found the pattern we were looking for.
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $D$.

Otherwise all of them are 0-valent.
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $D$

Then consider $C_1$ a configuration in $C$ reachable from $C$ that leads to $D_1$ a 1-valent configuration in $D$ by applying event $e$
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $D$.

Since event $e$ is applicable from $C$ then one can apply this event all along the path from $C$ to $C_1$. 

\[ e = (m, p) \]
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $\mathcal{D}$.

All these configurations belong to $\mathcal{D}$. Hence they are all univalent. Some of them can be 1-valent as is $D_1$. 

![Diagram](image-url)
\( \mathcal{C} \) contains two neighbor configurations \( C_0 \) and \( C_1 \) that lead to 0-valent \( D_0 \) and 1-valent \( D_1 \) in \( D \).

If one of them is 0-valent, we are done. We have found the pattern we were looking for.
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $D$.

Otherwise all of them are 1-valent.
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $\mathcal{D}$

The pattern we are looking for is located at configuration $C$. Let us apply step $e$ to $C$
\( \mathcal{C} \) contains two neighbor configurations \( C_0 \) and \( C_1 \) that lead to 0-valent \( D_0 \) and 1-valent \( D_1 \) in \( \mathcal{D} \)

Either this configuration of \( \mathcal{D} \) is 0-valent, and thus we can identify the pattern we were looking for.
$C$ contains two neighbor configurations $C_0$ and $C_1$ that lead to 0-valent $D_0$ and 1-valent $D_1$ in $D$.

Or this configuration of $D$ is 1-valent, and thus we can identify the pattern we were looking for.
Lemma 3

The proof is by contradiction

- We assume that every configuration $D \in \mathcal{D}$ is univalent
- Part 1: We have shown that $D \in \mathcal{D}$ contains both 0-valent and 1-valent configurations
- Part 2: We have shown that $\mathcal{C}$ contains two neighbor configurations $C_0$ and $C_1$ that lead to respectively a 0-valent and 1-valent configuration in $\mathcal{D}$
- Part 3: Based on these two neighbor configurations we are reaching a contradiction
Assume that $p \neq p'$
Reaching a contradiction

Case $p \neq p'$

- Since $p$ is different from $p'$ then events $e$ and $e'$ do not interact.
- Event $e'$ can be applied to configuration $D_0$.
- Thus $D_0.e' = D_1$ which closes the diamond pattern.
- But a 0-valent configuration cannot lead to a 1-valent configuration.
- Contradiction with the assumption of the lemma.
Case \( p = p' \)

- The diamond pattern does not apply
Case $p = p'$
Consider a deciding execution that can be applied to $C_0$ in which
- $p$ does not make any receive step
- Let $\sigma$ be the corresponding sequence of events
- Let $A = C_0.\sigma$ be the decision configuration
Reaching a contradiction

Case $p = p'$
Since $p$ takes no step in $\sigma$, $\sigma$ can be applied to $D_0$ and to $D_1$
Reaching a contradiction

Case $p = p'$
Leading to a 0-valent configuration $E_0$ and 1-valent configuration $E_1$
Case $p = p'$
Now the adversary allows $p$ to make its step $e$ from configuration $A$. This leads to configuration $E_0 = e(A)$. 
Reaching a contradiction

Case $p = p'$
Both $e'$ and $e$ can be applied to configuration $A$ and leads to $E_1 = e(e'(A))$. 

\[
\begin{align*}
\sigma_A & \quad E_0 
\sigma & \quad C_0 \\
D_0 & \quad 0\text{-valent} & e = (p, m) \\
C_1 & \quad e = (p, m) \\
D_1 & \quad 1\text{-valent} \\
E_1 & \quad 1\text{-valent} \\
C_A & \quad e' = (p, m') \\
A & \quad e = (p, m) \\
E_0 & \quad 0\text{-valent}
\end{align*}
\]
Case $p = p'$

- Thus $A$ must be bivalent, which is impossible since $\sigma$ is a deciding sequence of events (i.e., $A$ is univalent)
- Contradiction with the assumption
Bridging it all together

- Any deciding execution from a bivalent configuration goes to an univalent configuration, so there must exist some event that goes from a bivalent configuration to a univalent one. Such an event determines the decision value.
Conclusion on the Consensus abstraction

- Solutions exist in synchronous systems (even with Byzantine failures)
- Impossible to solve in an asynchronous system
- This theorem is so far the most fundamental one for the field of fault-tolerant distributed computing
- To circumvent this impossibility result, one can use
  - Randomization
    - At some point of the algorithm, i.e., when processors do not know what to decide, they can use randomization
  - Failure detectors
    - Oracles that give some hints on processes state of failure
Any questions?