Concensus for Asynchronous Systems

Etienne Renault

2 octobre 2020

https://www.lrde.epita.fr/~renault/teaching/algorep/

FLP [1]

Abstract of the paper

The **consensus problem** involves an **asynchronous system of processes**,some of which **may be unreliable**. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that **every protocol for this problem has the possibility of nontermination**, even with only one faulty process.

1. *Impossibility of Distributed Consensuswith One Faulty Process*, 1985, Fischer, Lynch and Paterson

FLP [1]

Abstract of the paper

The **consensus problem** involves an **asynchronous system of processes**,some of which **may be unreliable**. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that **every protocol for this problem has the possibility of nontermination**, even with only one faulty process.

Impossibility Result

No completely asynchronous consensus protocol can tolerate even a single unannounced process death.

1. *Impossibility of Distributed Consensuswith One Faulty Process*, 1985, Fischer, Lynch and Paterson

Problem Description

- Every process starts with an initial value in $\{0, 1\}$;

Problem Description

- Every process starts with an initial value in $\{0, 1\}$;
- **Agreement** : A nonfaulty process decides on a value, and enters a decision state

Problem Description

- Every process starts with an initial value in $\{0, 1\}$;
- **Agreement** : A nonfaulty process decides on a value, and enters a decision state
- **Termination :** All nonfaulty processes are required to choose the same value

Problem Description

- Every process starts with an initial value in $\{0, 1\}$;
- **Agreement** : A nonfaulty process decides on a value, and enters a decision state
- **Termination :** All nonfaulty processes are required to choose the same value
- Every message is eventually delivered as long as the destination makes infinitely many attempts to receive, but messages can be delayed arbitrarily long and delivered out of order

Problem Description

- Every process starts with an initial value in $\{0, 1\}$;
- **Agreement** : A nonfaulty process decides on a value, and enters a decision state
- **Termination :** All nonfaulty processes are required to choose the same value
- Every message is eventually delivered as long as the destination makes infinitely many attempts to receive, but messages can be delayed arbitrarily long and delivered out of order

The result of the consensus algorithm is predetermined only by the initial configuration.

Configurations

A configuration is defined as the internal state of all of the processes with the contents of the message buffer.

- **0-valent** configuration can only lead to choose 0
- **1-valent** configuration can only lead to choose 1
- **bi-valent** configuration can lead to choose 0 or 1

Proof.

Show circumstances under which the protocol remains forever indecisive

Proof.

Show circumstances under which the protocol remains forever indecisive

Intuition

If you delay a message that is pending any amount from one event to arbitrarily many, there will be one configuration in which you receive that message and end up in a bivalent state.

Proof.

Show circumstances under which the protocol
remains forever indecisive

Intuition

If you delay a message that is pending any amount from one event to arbitrarily many, there will be one configuration in which you receive that message and end up in a bivalent state.

Proof.

Show circumstances under which the protocol remains forever indecisive

Intuition

If you delay a message that is pending any amount from one event to arbitrarily many, there will be one configuration in which you receive that message and end up in a bivalent state.

We need two Lemma :

1. There is some initial configuration in which the decision is not predetermined, but in fact arrived as a result of the sequence of steps taken and the occurrence of any failure
2. If you delay a message that is pending any amount from one event to arbitrarily many, there will be one configuration in which you receive that message and end up in a bivalent state.

First Lemma 1/2

Theorem

The protocol P has a bivalent initial configuration

Proof.

- Suppose that the opposite was true that all initial configurations have predetermined executions

First Lemma 1/2

Theorem

The protocol P has a bivalent initial configuration

Proof.

- Suppose that the opposite was true that all initial configurations have predetermined executions
- Each configuration is uniquely determined by the set of initial values in the processes

First Lemma 1/2

Theorem

The protocol P has a bivalent initial configuration

Proof.

- Suppose that the opposite was true that all initial configurations have predetermined executions
- Each configuration is uniquely determined by the set of initial values in the processes
- Suppose that we have one configuration that is 0-valent (C0) and one that is 1-valent (C1)

First Lemma 1/2

Theorem

The protocol P has a bivalent initial configuration

Proof.

- Suppose that the opposite was true that all initial configurations have predetermined executions
- Each configuration is uniquely determined by the set of initial values in the processes
- Suppose that we have one configuration that is 0-valent (C0) and one that is 1-valent (C1)
- From C0 there must be a run that decides 0 even if p fails initially

First Lemma 2/2

Proof cont'd.

- Therefore p neither sends nor receives any messages, so its initial value cannot be observed by the rest of the processors.

First Lemma 2/2

Proof cont'd.

- Therefore p neither sends nor receives any messages, so its initial value cannot be observed by the rest of the processors.
- One of whom must eventually decide 0

First Lemma 2/2

Proof cont'd.

- Therefore p neither sends nor receives any messages, so its initial value cannot be observed by the rest of the processors.
- One of whom must eventually decide 0
- This run can also be made from C1.

First Lemma 2/2

Proof cont'd.

- Therefore p neither sends nor receives any messages, so its initial value cannot be observed by the rest of the processors.
- One of whom must eventually decide 0
- This run can also be made from C1.
- So one process must eventually decide 1

First Lemma 2/2

Proof cont'd.

- Therefore p neither sends nor receives any messages, so its initial value cannot be observed by the rest of the processors.
- One of whom must eventually decide 0
- This run can also be made from C1.
- So one process must eventually decide 1
- This contradicts our assumption that the result of the consensus algorithm is predetermined only by the initial configuration.

Second Lemma 1/2

- Let C be a configuration and e some event applicable to C.

Second Lemma 1/2

- Let C be a configuration and e some event applicable to C.
- Let $\mathbb{C}$ be the set of configurations reachable from C without applying e

Second Lemma 1/2

- Let C be a configuration and e some event applicable to C.
- Let $\mathbb{C}$ be the set of configurations reachable from C without applying e
- Let $\mathbb{D}$ be the set of configurations resulting from applying e to configurations in $\mathbb{C}$

Second Lemma 1/2

- Let C be a configuration and e some event applicable to C.
- Let $\mathbb{C}$ be the set of configurations reachable from C without applying e
- Let $\mathbb{D}$ be the set of configurations resulting from applying e to configurations in $\mathbb{C}$

Theorem

$\mathbb{D}$ contains a bivalent configuration.

Proof.

- Assume that $\mathbb{D}$ contains no bivalent configurations.
- If $\mathbb{D}$ is univalent, then $\mathbb{C}$ should be univalent since any configuration in $\mathbb{C}$ can reach a configuration in $\mathbb{D}$
- By Contraction $\mathbb{D}$ contains a bivalent configuration.

Altogether

- Start from a bivalent initial configuration C0

Altogether

- Start from a bivalent initial configuration C0
- To make the run admissible, place the processes in the system in a queue and have them receive messages in queue order, being placed at the back of the queue when they are done. This ensures that every message is eventually delivered.

Altogether

- Start from a bivalent initial configuration C0
- To make the run admissible, place the processes in the system in a queue and have them receive messages in queue order, being placed at the back of the queue when they are done. This ensures that every message is eventually delivered.
- Let *e* be the earliest message to the first processor in the queue, possibly null

Altogether

- Start from a bivalent initial configuration C0
- To make the run admissible, place the processes in the system in a queue and have them receive messages in queue order, being placed at the back of the queue when they are done. This ensures that every message is eventually delivered.
- Let e be the earliest message to the first processor in the queue, possibly null
- Then by the second lemma we can reach a bivalent configuration C1 reachable from C0 where e is the last message received.

Altogether

- Start from a bivalent initial configuration C0
- To make the run admissible, place the processes in the system in a queue and have them receive messages in queue order, being placed at the back of the queue when they are done. This ensures that every message is eventually delivered.
- Let e be the earliest message to the first processor in the queue, possibly null
- Then by the second lemma we can reach a bivalent configuration C1 reachable from C0 where e is the last message received.
- Similarly, we can reach another bivalent configuration C2 from C1 by the same argument. And this may continue for ever.