Impossibility of Distributed Consensus with One Faulty Process *

Antonio Franco

March 3, 2015

*Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1983. Impossibility of distributed consensus with one faulty process. In Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems (PODS '83). ACM, New York, NY, USA, 1-7. DOI=10.1145/588058.588060 http://doi.acm.org/10.1145/588058.588060

Introduction

- Problem: reaching agreement among remote processes (You can think of consensus as “voting” for a value.);
- Example: Different processes have to decide wether to insert a value into a database, result of a transaction they concurred to process, or not, in order to preserve the database consistency;
- In case of failure of one or more processes, or, in the extreme case, injection of one or more malicious processes, the question is wether or not is possible to ensure a certain grade of ”resilience“, or ensure, at least, a tolerance to a certain number of faults;
- The result is quite surprising: **No completely asynchronous consensus protocol can tolerate even a single unannounced process death**.

Preview of the results

- The results are quite surprising: **No completely asynchronous consensus protocol can tolerate even a single unannounced process death**.
- Even in the absence of Byzantine processes (from now on an assumption), the stopping of a single process at an inopportune time can cause any distributed commit protocol to fail to reach agreement;
- This confirmed what was believed at the time, that asynchronous commit protocols have a "window of vulnerability" i.e. an interval of time during the execution of the algorithm in which the delay or inaccesibility of a single process can cause an infinite loop in the entire algorithm.

Assumptions I

- Every process starts with an initial value in $\{0, 1\}$;
- A nonfaulty process decides on a value, and enters a *decision state*;
- All nonfaulty processes are required to choose the same value;
- Only some of the processes are required to choose the same value (in order to broaden the scope of the proof);
- Both 0 and 1 are possible decision values;
- Processes are modeled as automata, which communicate by meaning of messages;
- The atomic step for a process is:
  - attempt to receive a message
  - perform local computation based on wether or not a message was delivered to it
  - send a finite set of messages to other processes as a broadcast.

Assumptions II

- 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.

Model definition I

- $N \geq 2$ processors which communicate by sending messages;
- A message is a pair (p,m) where p is the processor the message is intended for, and m is the contents of the message.
- Message buffer: multiset storing messages;
  - buffer.send(p,m) places the message (p,m) in the message buffer;
  - buffer.receive(p) either returns a message for processor p (and removes it from the message buffer) or the special value θ, which does nothing;
- A configuration is defined as the internal state of all of the processors – the current step in the algorithm that they are executing and the contents of their memory – together with the contents of the message buffer.
- The system moves from one configuration to the next by a step which consists of a processor p performing receive(p) and moving to another internal state.

Model definition II

- A particular execution, defined by a possibly infinite sequence of events from a starting configuration C is called a schedule and the sequence of steps taken to realise the schedule is a run.
- We say that a run is deciding provided that some process eventually decides according to the properties of consensus, and that a consensus protocol is totally correct if every admissible run is a deciding run.

Proof

- The basic idea is to show circumstances under which the protocol remains forever indecisive;
- We call the configurations that may lead to either decision value bivalent, and configurations that will only result in one value 0-valent or 1-valent.
- Two main lemmas:
  1. Show that 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 I

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 processors.
- We can order these values in a chain where two configurations are next to each other if they differ by only one value, so the only difference between two adjacent configurations is the starting value of one processor.
- At some point along this chain, a 0-deciding initial configuration must be next to a 1-deciding one, and therefore they must differ by only one initial value.

First lemma II

- Call the 0-deciding configuration C_0 and the 1-deciding configuration C_1. Call the processor whose initial value is different in both p.
- Now, from C_0 there must be a run that decides 0 even if p fails initially – because we allow one processor to fail.
- 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 C_1. Because the configurations of C_0 and C_1 are exactly the same, except for the value at p which can't contribute anything to the run because p has failed, any sequence of steps that C_0 may take can also be taken by C_1.
- But if they take exactly the same sequence of steps, then the eventual decision taken must be the same for both configurations.

First lemma III

- This contradicts our assumption that the result of the consensus algorithm is predetermined only by the initial configuration.
- So either one (or possibly both) of these configurations can potentially decide 0 or 1 and **the eventual result depends on the pattern of failures and message deliveries**.

Second lemma I

Let C be a bivalent configuration and let e=(p,m) be some event that is applicable to C. Let $\mathbb{C}$ be the set of configurations reachable from C without applying e, and 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.

Second lemma II

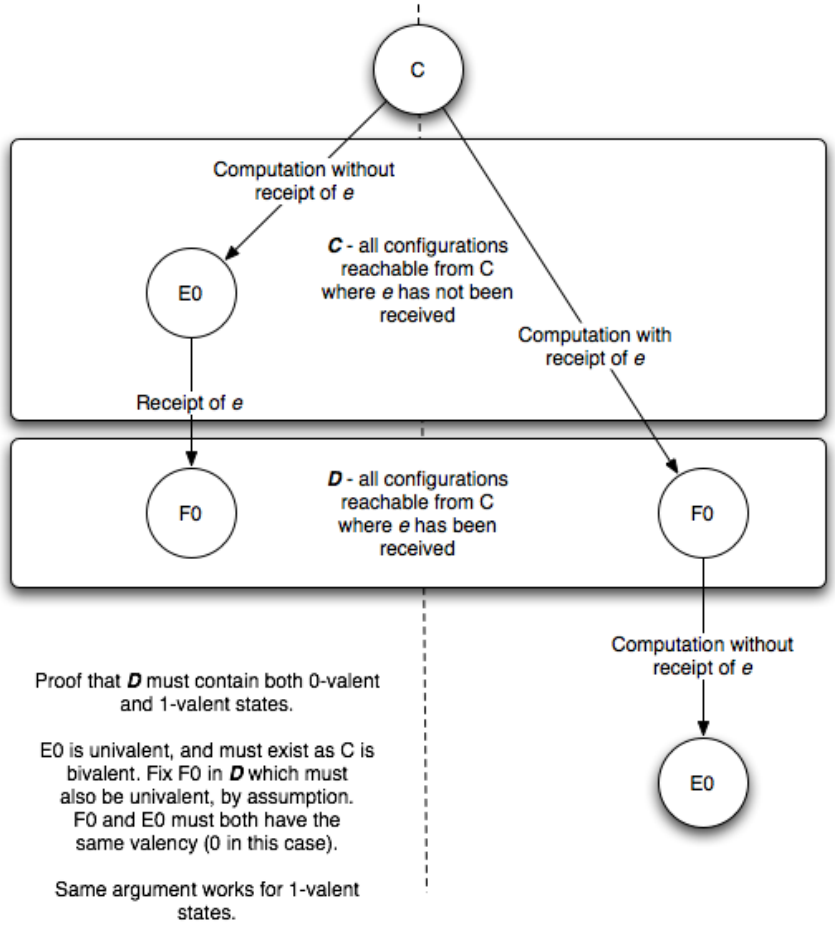

Figure : Proof that D must contain both 0- and 1-valent configurations if it contains no bivalent configuration (`http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/`)

Second lemma III

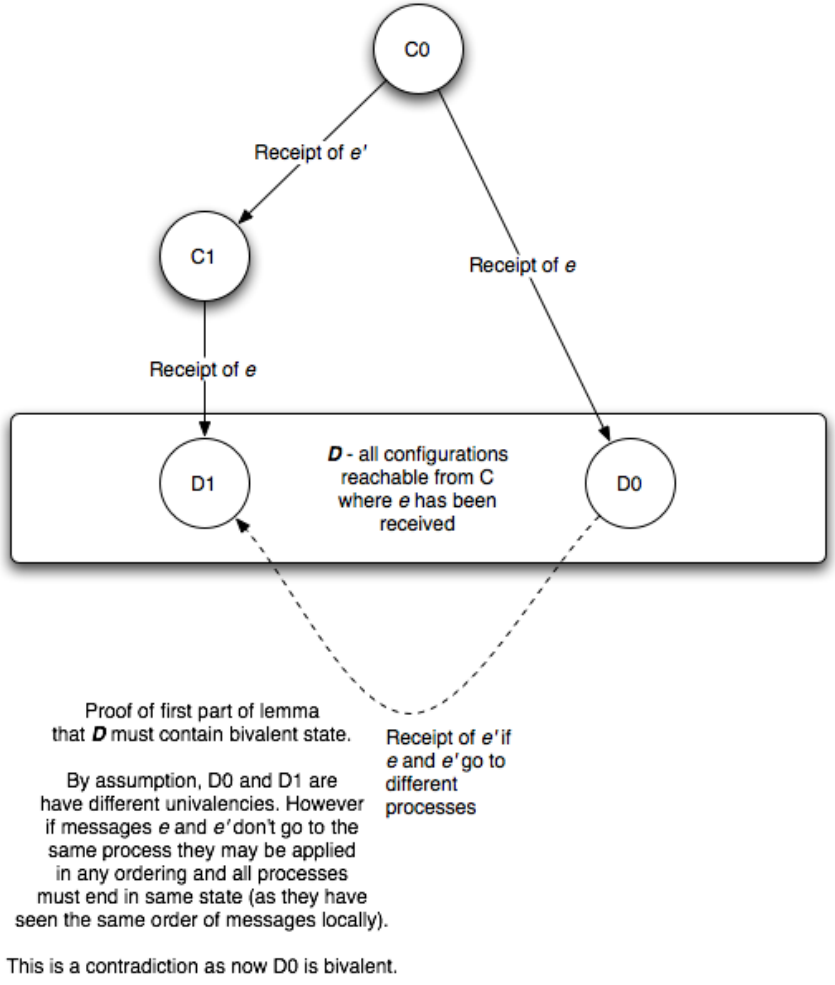

Figure : Proof that D_0 must be bivalent if the two states preceding D_1 are separated by messages intended for different processes; C_0 and C_1 are adjacent neighbours; e=(p,m), e'=(p',m') and $p' \neq p$. (`http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/`)

Second lemma IV

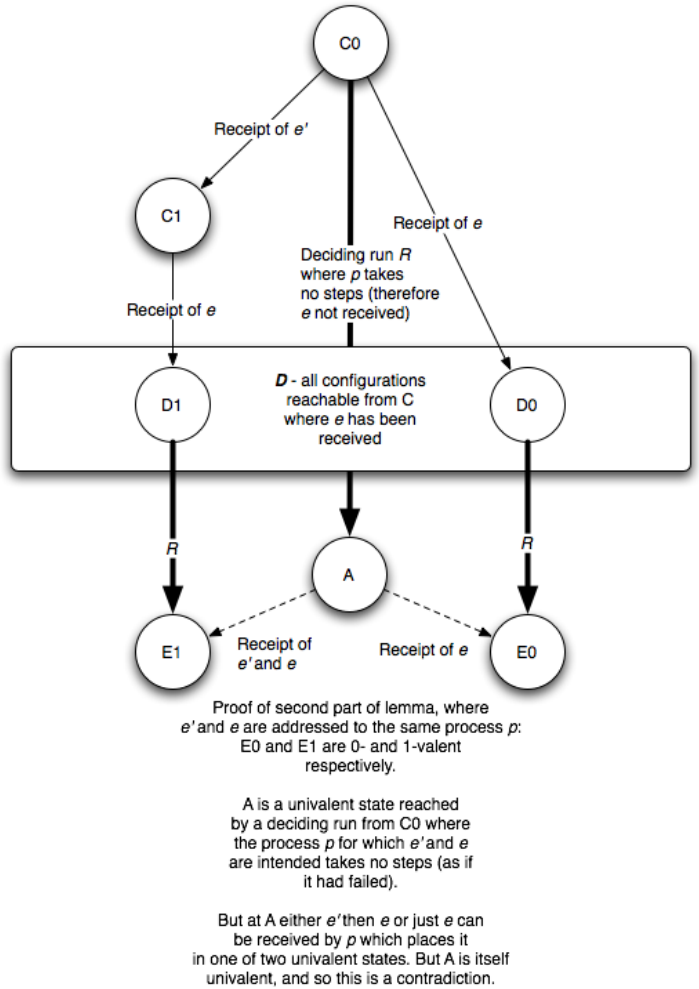

Figure : Proof that no deciding run from C0 exists if D contains only univalent configurations, and both messages preceding D1 are delivered to the same host (i.e. $p' = p$).
(`http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/`)

Second lemma V

- So, by contradiction, we have shown that $\mathbb{D}$ must contain a bivalent configuration.

□

Lemmas together I

- The final step is to show that any deciding run also allows the construction of an infinite non-deciding run.
- Start from a bivalent initial configuration C_0 (which exists as a result of the first lemma).
- To make the run admissible, place the processors 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=(p,m) be the earliest message to the first processor in the queue; possibly a null message.
- Then by the second lemma we can reach a bivalent configuration C_1 reachable from C_0 where e is the last message received.
- Similarly, we can reach another bivalent configuration C_2 from C_1 by the same argument. And this may continue for ever.

Lemmas together II

- This run is admissible, but never reaches a decision, and it follows that the protocol is not totally correct. □

Further reading

- The original paper: Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1983. Impossibility of distributed consensus with one faulty process. In Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems (PODS '83). ACM, New York, NY, USA, 1-7. DOI=10.1145/588058.588060 `http://doi.acm.org/10.1145/588058.588060`;
- Good walkthrough: `http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/`;