Impossibility of Distributed Asynchronous Consensus with One Faulty Process

by

Niu Liu

Introduction

Consensus agreement with fault tolerance is a fundamental topic in distributed systems. The purpose of the topic is members of a distributed systems wish to adopt a consistent decision even under cases that there might exist some fault nodes. The theory result has great affection on the design of distributed systems. However, this topic is complex for several reasons. First, we should have very clear definitions for terminologies evolved, a clear description of context. Different definition and context will lead to different result. Second, the background logic process to proof is also profound and subtle. In this essay, we will discuss the famous theorem of **FLP Impossibility**, the result is **in a asynchronous environment, even there is only one faulty process, there is no such an algorithm to make sure all non-faulty processes to achieve an valid agreement**.

Preliminary of terminologies and assumptions

To discuss the FLP result, we should give clear some definitions and assumptions first.
To get a valid consensus for processors, the algorithm should achieve the three goals:

- **Termination:** All non-faulty processes decide on a value eventually.
- **Agreement:** All non-faulty processes should decide on the same value.
- **Validity:** the final value decided must be proposed by at least one process.[2]

Termination means the algorithm cannot run without end. All non-faulty processes should finally decide on a value in a finite time. Agreement is also self explanatory. If there are two non-faulty processes decide on a different value, it is a failure. The validity is a bit difficult to understand at the first glance. The requirement of validity is to get a reasonable decision, prevent trivial solutions. For example, the solution of “all non-faulty processes decide on value 0”. This solution satisfies termination and agreement, but it is useless in reality.

Other definitions are:
Asynchronous: the asynchronous means for any process, there is no upper bound on how long it would take to complete a task. We cannot use timeout to decide that this process is crashed or not.[1]

Fail-stop: Processes fail to execute any works or crashed.

Byzantine failure: Process with byzantine failure means it might be hacked with malicious software, it can take any possible actions to send normal or error messages to others.[4]

State machine: each process has a current state, including the contents of the memory and message buffer, the current step of the algorithm it execute. Each time a state machine receive a new message or execute a step of the algorithm, it will change to another state, possibly output a message. [3]

Configuration: a configuration is the state of all processes. The only way that a system state change is when some processes receive a message.[2]

Deterministic system: for a given initial state, a system always produce the same output.

Non-deterministic: a system cannot guarantee to produce the same output based on a given initial state.

We need to provide the assumptions.

- **Failure of processes:** processes are allow to fail for fail-stop or byzantine failure.
- **Reliability of connections**: we assume the links (Logical Link) between each pairs of processes are reliable.
- **Asynchronous:** the distributed system is asynchronous

Sketch of the Impossibility of theorem

Now based on the terminologies and assumptions, we describe the problem in a formal way:

Impossibility theorem

In an **asynchronous** environment, for a distributed system including N (N >= 2) processes, even there is only one faulty process, no such a consensus algorithm which satisfies **termination**, **agreement** and **validity** exists for all non-faulty processes.

Sketch of the proof

In fact, the strict proof of this theorem is complicated and tedious, so here I would not like to provide the whole proof, but give a tour description to help understanding it.

First, let's prove a lemma, which is much simpler and is useful to lead to the result of the Impossibility theorem.

Non-deterministic lemma
In an asynchronous environment, assuming the system can tolerate one faulty-process, there exists some initial configurations such that the decision is non-deterministic. [1]

It means the final decision does not only depend on the initial configuration, the execute step and execute time of some processes will also affect the final decision.

To prove this lemma, we suppose the opposite is true - there is such an algorithm, under which all initial configurations can determine the final decision.

According to validity of the algorithm, it must be true that there are some initial configurations which result in "0" and some initial configurations which result in "1". Each configuration is uniquely determined by the initial values of the processes. For configurations which result in "0", we call it "0" deciding configurations. For configurations which result in "1", we call it "1" deciding configurations.

Now we order all the configurations in a chain that any two configurations side by side differ by only one initial value. There must be exist a point where a "0" deciding initial configuration is next to a "1" deciding initial configuration. We name this two configurations C0 and C1. C0 and C1 are different only by one process P.

Based on the assumption that the algorithm can tolerate one faulty process and C0 is a "0" deciding configuration, we can deduce from C0 the final decision will be "0" even if P fails. For the same reason, we can also get the result that from C1 the final decision will be "1" even if P fails. But we know C0 and C1 only differ by P. in case that P fails, we get the that result must be "0" and also the result must be "1", this is a contradiction. So the opposite of the lemma is not true, the lemma is proved.[2]

Based on the lemma, it would be easy to understand the **Impossibility theorem**. Since there exists some configurations (C0 or C1) from which the final decision is not deterministic, the final decision will be affected by the execution of process P. The P might be fail to respond. Now the system will be in an endless state. Since the system is fault tolerant for one fault process, if P crashed, the other non-faulty process should move on to reach a decision. But before the final decision is reached, process P responds with a message, all the non-faulty processes had to adopt P's message in decision, then the system is back to the initial step. This would be an

endless process to get a final decision, which does not satisfy the **termination** requirement of the consensus algorithm. So, such an algorithm does not exist.

Further thinking

FLP Impossibility and Byzantine Agreement

Let's compare FLP Impossibility with Byzantine Agreement. In Byzantine Agreement, for an N nodes system, Byzantine agreement can tolerate faulty-processes if the number of faulty processes are less than 3/N. While FLP Impossibility gets the result that the system cannot tolerate even one faulty process, what is the root difference ?

If we compare the context and assumptions we can get the root reason, the core difference between these two is, in FLP Impossibility, we assume the system is **asynchronous**, while in Byzantine Agreement, this assumption is not true. Let's describe it in more detail.

Asynchronous assumption means there is no upper bound on how long a process will take to execute a task, other process cannot use timeout to determine that this process is crashed. While in Byzantine Agreement, processes use timeout as a method to determine that some processes are crashed. This is the core difference and the core reason why FLP impossibility and Byzantine agreement get a different result.

FLP Impossibility and Paxos Algorithm

For Paxos Algorithm, the assumption of it is

1. **Asynchronous**
2. Network is reliable but maybe slow

Since we know that Paxos Algorithm is fault tolerance, is it contradict with the FLP impossibility ?

The Answer is no, because Paxos Algorithm cannot guarantee **liveness and termination** [3], it can only get liveness as good as it can. While what FLP impossibility describes is no algorithm can satisfy **termination**, agreement, and validity with fault tolerance for even one faulty-process. So Paxos and FLP impossibility are not contradiction.

References:

[1] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. In *Proceedings of the 2nd Annual ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (Atlanta, Ga., Mar. 21-23). Retrieved from http://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf.

[2] A Brief Tour of FLP Impossibility. In Computer Systems, *Distributed Algorithms and Databases.* August 13, 2008. Retrieved from http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/.

[3] Xinfeng Ye. The consensus problem in distributed systems. Retrieved from https://canvas.auckland.ac.nz/courses/14774/files/479601.

[4] Radu Nicolescu. The Byzantine Agreement - An Introduction. Retrieved from https://www.cs.auckland.ac.nz/courses/compsci711s2c/lectures/radu/04-Byzantine-handout.pdf.