A simple and efficient randomized byzantine agreement algorithm

Filip Jasionowicz

January 2024

Abstract

The goal of this project is to implement a byzantine algorithm proposed by Benny Chor and Brian A. Coan [1] as well as provide a sketch of the correctness analysis.

1 Introduction

1.1 Problem formulation

Goal of the algorithm is to produce an agreement (consensus) on one bit between non-faulty processors having initial preferences $v_1, ..., v_n$. We want agreed value to fulfill the following conditions:

- it's the same for all non-faulty processors
- if $v_1 = v_2 = ... = v_n$ then the decided value is also v_1

1.2 Model of computation

Implemented algorithm operates in the following model:

- System is synchronous
- Each processor can communicate with any other in the network and connections never fail
- All non-faulty processors run the same code
- Each processor knows the total number of processors n and maximum number of faulty processors t
- There are no limitations on the behaviour of the faulty processors, in particular they can try to intentionally disrupt the agreement

Before presenting the algorithm it is necessary to make some assumptions about the value of t. Lamport et al. have proven in [2] that for the correct deterministic solution to exist we need $n \geq 3t + 1$. Since their reasoning can be simply translated into the non-deterministic case we will assume from this point that $n \geq 3t + 1$. For the sake of simplicity in the following reasoning we will assume that any bit can either have a regular value (0 or 1) or unknown/undecided value which we denote by "?". This assumption makes algorithm easier to read at the same time being trivial to implement.

2 The algorithm

The following pseudocode is intended to run on each non-faulty processor:

Algorithm 1 Byzantine agreement

1: $curr \leftarrow$ **input**
2: $toss \leftarrow$ "?"
3: $epoch \leftarrow 0$
4: **repeat**
5: **broadcast**$(curr)$ ▷ round one
6: **receive_all**(v_{curr})
7: **if** for some bit b we got $\geq n - t$ messages containing b **then**
8: $curr \leftarrow b$
9: **else**
10: $curr \leftarrow$ "?"
11: **if** $my\_group\_id \equiv epoch \mod \left\lfloor \frac{n}{g} \right\rfloor$ **then**
12: $toss \leftarrow$ **toss_coin()**
13: **else**
14: $toss \leftarrow$ "?"
15:
16: **broadcast**$(curr, toss)$ ▷ round two
17: **receive_all**(v_{curr}, v_{toss})
18: $ans \leftarrow$ the bit $b \neq$ "?" such that $(b, *)$ messages are most frequent
19: $num \leftarrow$ number of ocurrences of $(ans, *)$ messages
20: **if** $num \geq n - t$ **then**
21: **decide** ans ▷ Agreement reached with ans as value
22: **return**
23: **else if** $num \geq t + 1$ **then**
24: $curr \leftarrow ans$
25: **else**
26: $curr \leftarrow$ the bit $b \neq$ "?" such that $(*, b)$ messages are most frequent
27: $epoch \leftarrow epoch + 1$
28: **until** agreement reached

Algorithm is divided into phases, each of them consisting two rounds. Phases

are conducted until agreement is reached. Processors are divided into $\left\lfloor \frac{n}{g} \right\rfloor$ groups of size g. The value of g is set to be around $\log(n)$ for the reasons provided in the section 3.

In turn one *curr* value is broadcasted and updated accordingly to received messages. After that, processors from the group which *id* is corresponding to *epoch* modulo $\left\lfloor \frac{n}{g} \right\rfloor$ perform a coin toss - uniformly at random select 0 or 1.

In turn two *curr* and *toss* values are broadcasted and *curr* is updated accordingly to received messages. Additionally we determine whether consensus has been reached. If not - we continue the algorithm and perform next epoch. Otherwise we finish.

3 Sketch of analysis

3.1 Partial correctness

First, let's notice that if all processors get the same input v, they automatically decide on it in one phase since in round one all($\geq n - t$) non-faulty processors send v and in round two they accept it.

Lemma 1. *At most one value (by value we understand 0 or 1, "?" is not a value) is sent in round 2 by non-faulty processors.*

Proof. When a processor sends a non-"?" value v, it must have been gotten in round one from at least $n-t$ processors of which at least $n-2t$ were non-faulty. This means that in round one each processor has gotten at least $n-2t$ values v, thus at most $2t < n - t$ messages contained a value other than v so it couldn't be chosen by any other non-faulty processor. □

To decide on v in round two processors need to see at least $n - t$ messages containing v. At least $n - 2t \geq t + 1$ of those messages are from non-faulty processors, since by Lemma 1 only faulty processors could have sent values different than v. All of non-faulty processors either immediately decide on value or save it as *curr* and decide on it in next round.

3.2 Termination

By Lemma 1 there is at most one value v broadcasted by a non-faulty processor. This implies, that each non-faulty processor either sets *curr* to v or looks at majority coin toss. Since there are not too many faulty processors, they can't decide the majority of coin tosses in all groups, so there is at least one group in which both values can be selected as a majority coin toss (although probability distribution will not necessarily be uniform). After some computation we can conclude that for group size $g = \log(n)$ the expected number of epochs to reach agreement is bounded by $O\left(\frac{t}{\log(n)}\right)$.

References

[1] Benny Chor and Brian A Coan. A simple and efficient randomized byzantine agreement algorithm. *IEEE Transactions on Software Engineering*, (6):531–539, 1985.

[2] Marshall Pease, Robert Shostak, and L Lamport. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.