Byzantine Fault Allowing Protocols

LIAM MONNINGER, Ramate LLC, USA

This memo is for friends and colleagues to explain the beginnings of my study of a category of distributed computing protocols which I call Byzantine fault-allowing. Below, I will begin with a motivation for the study. I'll then provide a trivial combinatorial example of a protocol assumed to be in the category. Finally, I'll suggest more abstract constructions with which the study is properly concerned.

Most terms and notation follow [HKR]. The final section introduces provisional and more suggestive terms.

1 Motivation

The motivation for studying Byzantine fault-allowing protocols, as I have dubbed them, is derived primarily from two phenomena in the world of distributed computing: the need for fault-tolerant order and compute inefficiencies of Byzantine fault-tolerant replica-based protocols.

1.1 The General Need for Order

In computing, we typically reason about partial functions. These are functions which deterministically map one value to another, but may not be defined for all inputs.

$$\hat{f} : S \to S \cup \{\bot\} \text{ is a lifted partial function.}$$
$$f : S \rightharpoonup S \text{ is a partial function in its native category.}$$

When composing these partial functions, we axiomatize that undefinedness propagates through the composition.

$$f, g : S \rightharpoonup S$$
$$f \circ g \text{ is defined if and only if}$$
$$g(x) \text{ is defined and}$$
$$f(g(x)) \text{ is defined}$$

We find this to be a suitable model for modern machines, because we know how to design hardware which can logically represent functions with these properties and which is rarely corrupted by the surrounding environment.

However, for general computing tasks, we often have partial functions which are not commutative.

$$f \circ g \neq g \circ f$$

A familiar example is subtraction on $\mathbb{N}$, which is naturally partial:

$$\text{sub} : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$$

$$\text{sub}(a, b) = \begin{cases} a - b & \text{if } a \geq b, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

And is also not commutative:

$$\text{sub}(5, 3) = 2 \neq \text{sub}(3, 5) \text{ (undefined).}$$

In general, for a family $f_1, \ldots, f_n$ of partial functions, the composite is determined by the ordered sequence $(f_1, \ldots, f_n)$ not the unordered set $\{f_1, \ldots, f_n\}$. In local hardware cases—such as a single hart—we can often form reasonable assumptions or conventions, such that something like a state machine reading through an assembly is equivalent to a sequence of partial functions. In the context of general distributed computing tasks, however, we tend to make these assumptions more minimal.

In distributed computing, we may instead refer to a set of processes P which are able to share information with one another via messages $m \in M$ to varying degrees of success. Then, within this context, we hope to achieve results up to some notion of correctness and consistency, i.e., up to some notion of equivalence with the local hardware case.

Put more provocatively, a distributed computing protocol often has to decide from a chaotic world of messages what a reasonable order of partial functions should be. Signatures can be placed on messages to ensure they are authentic. Prover systems can be built to attest the correctness of the value of a partial function. But, these may mean little without a reasonable order.

1.2   2f + 1

Often the success of a distributed computing protocol is framed in terms of its agreement or consensus. And, in the abundant cases where order matters, this necessarily means agreement on said order.

If we take set of processes $|P| = 3f + 1$ and ask for the unique messages $M_i \subseteq M$ which indicate computing the partial function at index i, we may wonder what different requirements on the agreement of these messages might mean.

A simple majority $|M_i| > \frac{|P|}{2}$ would ensure by the Generalized Pigeonhole Principle that any two subsets of messages M_i, M_j would intersect in at least one process. In a sense, at least one process can attest to each ordering step. By recurrence, this reads out a total order.

Let $Q : M \rightarrow 2^P$ be the function which maps a message to the set of processes which have sent it. Then, we may define order as $Order(M_i)$ as follows:

$$Q(M_i) = \bigsqcup_{m \in M_i} Q(m)$$
$$O(M_i, M_j) := (Q(M_i) \cap Q(M_j) \neq \emptyset)$$
$$Order(M_i) = Order(M_{i-1}) \wedge C(M_{i-1}, M_i)$$
$$= Order(M_0) \wedge O(M_0, M_1) \wedge \cdots \wedge O(M_{i-1}, M_i)$$

Importantly, this simple majority model does not account for the possibility that a process may fault. As it turns out, the most aggressive assumption we can make and still retain the

possibility of these intersecting sets amongst non-faulty processes is that at most f processes may fault. This is the Byzantine fault model.

Let H be the set of honest processes and F be the set of faulty processes. Two quora must then intersect in at least $f + 1$ processes:

$$
\begin{aligned}
|Q(M_1) \cap Q(M_2)| &= \\
&|Q(M_1)| + |Q(M_2)| - |Q(M_1) \cup Q(M_2)| \\
&\geq (2f + 1) + (2f + 1) - (3f + 1) \\
&= f + 1
\end{aligned}
$$

We may then use this fact to compute the intersection of two subsets of messages M_1, M_2 in the presence of faulty processes:

$$
\begin{aligned}
|Q(M_1) \cap Q(M_2)| \\
&= |Q(M_1) \cap Q(M_2) \cap H| + |Q(M_1) \cap Q(M_2) \cap F| \\
|Q(M_1) \cap Q(M_2) \cap H| \\
&= |Q(M_1) \cap Q(M_2)| - |Q(M_1) \cap Q(M_2) \cap F| \\
f \geq |Q(M_1) \cap Q(M_2) \cap F| \\
|Q(M_1) \cap Q(M_2) \cap H| \geq f + 1 - f = 1
\end{aligned}
$$

A straightforward interpretation of this fact renders the ubiquitous family of Byzantine fault-tolerant state machine replication protocols (BFT SMR). Without a prover system—which is common in practice owing to the computational expense of systems like ZKPs—BFT SMRs requires at least $2f + 1$ of the honest processes to be engaged in the computing and broadcasting a partial function $f_i, f_{i+1}, \cdots, f_n$. With prover systems, this requirement holds up to the task of ordering.

Roughly speaking, if a Byzantine fault-tolerant SMR protocol tolerates f faulty processes, then it uses at best $\frac{1}{2f+1}$ of the available non-faulty compute.

2 A Byzantine Fault-allowing Protocol

If we want to access more compute in a Byzantine fault-tolerant setting, an initial thought is to simply relax the requirement on order. Perhaps we can include fewer processes and allow our total order to fail with an acceptably low likelihood.

First, consider the following objective function which is minimized for M_i when $Order(M_i) = 1$:

$$
\begin{aligned}
Obj(M_i) = 0 &\iff Order(M_i) = 1 \\
Obj(M_i) = 1 &\iff Order(M_i) = 0
\end{aligned}
$$

2.1 The RESAMPLE Protocol

We define the following Byzantine fault-allowing protocol RESAMPLE:

- For a given index, i on M, pick a random subcommittee $K_i \subseteq P$ s.t. $|K_i| = 3k + 1$.
- Accept M_i if and only if $|Q_{K_i}(M_i)| \geq 2k + 1$. Otherwise, pick another random subcommittee and repeat.

Let...

$$f' = |K_i \cap F|, h' = |K_i \cap H|, f' + h' = |K_i| = 3k + 1$$

Observe the following possible outcomes for selection of a random subcommittee:

- $|K_i \cap H| \geq 2k + 1 \iff$ Divine $= D$, representing a subcommittee which has an honest supermajority which computes M_i correctly.
- $|K_i \cap H| \leq k \iff$ Corrupt $= C$, representing a subcommittee which has a dishonest supermajority and may compute M_i incorrectly.
- $k < |K_i \cap H| < 2k + 1 \iff$ Undecided $= U$, representing all other cases wherein neither an honest nor dishonest supermajority exists and may either compute M_i correctly or disagree internally and not render a supermajority.

2.2   The Loss of the RESAMPLE Protocol

Let $\mathcal{S}(f, k)$ represent the total number of ways to select the subcommittee without replacement:

$$\mathcal{S}(f, k) = \binom{3f + 1}{3k + 1}$$

The total number of ways to select a Corrupt subcommittee is:

$$\mathcal{S}_C(f, k) = \sum_{f'=2k+1}^{\min(3k+1, f)} \binom{f}{f'} \cdot \binom{2f + 1}{3k + 1 - f'}$$

The total number of ways to select a Divine subcommittee is:

$$\mathcal{S}_D(f, k) = \sum_{h'=2k+1}^{\min(3k+1, 2f+1)} \binom{2f + 1}{h'} \cdot \binom{f}{3k + 1 - h'}$$

All other outcomes are by definition Undecided, so the total number of ways to select an Undecided subcommittee is:

$$\mathcal{S}_U(f, k) = \mathcal{S}(f, k) - \mathcal{S}_C(f, k) - \mathcal{S}_D(f, k)$$

Let S represent the transient state in which the RESAMPLE is sampling, i.e., has just started or produced a Undecided subcommittee. Let G represent the absorbing state in which RESAMPLE makes a "good" decision to select a Divine subcommittee. Let B represent the absorbing state in which RESAMPLE makes a "bad" decision to select a Corrupt

subcommittee. The Markov Chain is then given by the state space Ω and transition matrix T.

$$\Omega = \{S, G, B\}$$

$$T = \begin{pmatrix} \Pr[U] & \Pr[D] & \Pr[C] \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The probability of the algorithm making a "bad" decision to select a Corrupt subcommittee $\Pr[B]$ is then:

Let $x := \Pr[$ eventually hitting B $|$ start in S $]$

$x = \Pr[C] \cdot 1 + \Pr[D] \cdot 0 + \Pr[U] \cdot x$

$x - \Pr[U] \cdot x = \Pr[C] \implies x \cdot (1 - \Pr[U]) = \Pr[C] \implies x = \dfrac{\Pr[C]}{1 - \Pr[U]}$

Observe that...

$1 - \Pr[U] = \Pr[D] + \Pr[C] \implies \Pr[B] = \dfrac{\Pr[C]}{\Pr[D] + \Pr[C]}$

Conversely, the probability of the algorithm making a "good" decision to select a Divine subcommittee $\Pr[G]$ is then:

$$\Pr[G] = 1 - \Pr[B] = 1 - \frac{\Pr[C]}{\Pr[D] + \Pr[C]} = \frac{\Pr[D]}{\Pr[D] + \Pr[C]}$$

As I will show, the probability $\Pr[C]$ drops off quickly for a fixed sampling ratio $\gamma = \frac{k}{f}$ (with, say, $k = \lfloor \gamma f \rfloor$). First observe that the number of faulty replicas in a sampled subcommittee follows a hypergeometric distribution:

$$X := |K_i \cap F| \sim \text{Hypergeometric}(N, K, n),$$

$$C \equiv \{X \geq 2k + 1\},$$

$$\Pr[C](\gamma, f) = \frac{\mathcal{S}_C(f, \lfloor \gamma f \rfloor)}{\mathcal{S}(f, \lfloor \gamma f \rfloor)}.$$

Standard large-deviation bounds for sampling without replacement give a KL-based tail bound:

$$\Pr[X \geq qn] \leq \exp(-n \cdot D(q \,\|\, p)),$$

$$D(q \,\|\, p) = q \log\left(\frac{q}{p}\right) + (1 - q) \log\left(\frac{1 - q}{1 - p}\right).$$

In our case $p = \frac{f}{3f+1} \approx \frac{1}{3}$, $q = \frac{2k+1}{3k+1} \approx \frac{2}{3}$, and $n = 3k + 1$. Hence

$$D\left(\frac{2}{3} \,\middle\|\, \frac{1}{3}\right) = \frac{2}{3} \log 2 + \frac{1}{3} \log \frac{1}{2} = \frac{1}{3} \log 2.$$

Therefore,

$$\Pr[C] \lesssim \exp\left(-(3k+1) \cdot \frac{1}{3} \log 2\right) = 2^{-(3k+1)/3},$$

$$\Pr[C](\gamma, f) \lesssim 2^{-(3\gamma f+1)/3} = 2^{-\gamma f - 1/3} = 2^{-\Omega(f)}.$$

In particular, for any fixed $\gamma \in (0, 1)$,

$$\lim_{f \to \infty} \Pr[C](\gamma, f) = 0 \forall \gamma \in (0, 1)$$

Meanwhile, the expect cost of the algorithm in terms of the number of processes used remains on the order $\Theta(k)$.

Thus, RESAMPLE is a Byzantine fault-allowing protocol which is computationally efficient and has a low probability of failure.

3   An Expanded View

Acknowledgments