Byzantine Fault Allowing Protocols

LIAM MONNINGER, Ramate LLC, USA

I've written this memo to describe to friends and colleagues the beginning of my study of a category of distributed computing protocols which I call Byzantine Fault Allowing. I begin with the motivation for the study. I then provide a trivial combinatorial example of a protocol assumed to be in the category. Finally, I 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

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.

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

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:

Author's Contact Information: Liam Monninger, liam@ramate.io, Ramate LLC, Durham, California, USA.

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

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 $Q : M \to 2^P$ 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:

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

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 $f + 1$ replicas:

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

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:

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

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 replicas 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 Simple Example of a Byzantine Fault-allowing Protocol

If we want to access more compute, an obvious thought is to simply relax the requirement–to allow our total order to potentially fail and perhaps prove such is acceptable.

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

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

One would quickly point out that since our partial functions can compute anything, it's best to assume that even a single failure on this requirement would be catastrophic. Thus, the likelihood of total order failing needs to be cosmologically low.

Consider the following simple example of a Byzantine fault-allowing protocol:

Acknowledgments