Crimes et Châtiments dans les Systèmes Distribués

Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, Zarko Milosevic, Adi Serendinschi

Crimes et Châtiments dans les Systèmes Distribués

Pierre Civit[1] et Seth Gilbert[2] et Vincent Gramoli[3,4] et Rachid Guerraoui [4] et Jovan Komatovic [4] et Zarko Milosevic [5] et Adi Serendinschi [5]

[1] *Sorbonne University, CNRS, LIP6*
[2] *National University of Singapore*
[3] *University of Sydney*
[4] *EPFL*
[5] *Informal Systems*

Considérons un *protocole distribué* non synchrone dont les n processus assurent des propriétés de *sûreté* et de *vivacité* malgré $t \leq n$ défaillances (byzantines) arbitraires. Malheureusement, on sait qu'il existe toujours une borne t_0 où il est impossible de garantir à la fois (i) la sûreté et la vivacité pour $t \leq t_0$ et (ii) la sûreté seulement pour $t > t_0$.

Dans cet article, nous proposons une transformation générique, appelée τ_{scr}, de tout protocole distribué en une version *responsable* (*accountable*) avec un factor $f = min(\lceil n/3 \rceil - 1, t_0)$ qui (i) continue à garantir la sûreté et la vivacité pour $t \leq f$ et (ii) garantit que les processus corrects obtiennent ultimement une preuve de culpabilité contre au moins $t_0 + 1$ processus byzantins responsables de toute violation de sûreté. Combinée à une punition adéquate, une telle détection incite naturellement à l'exemplarité.

Mots-clefs : responsabilité, détection, fautes Byzantines, incitatifs, systèmes distribués

1 Introduction

There are known limitations to the tasks distributed protocols can solve. For example it is well-known that the consensus problem cannot be solved if more than $t_0 = \lceil n/3 \rceil - 1$ processes are Byzantine. Similar results apply to set agreement or lattice agreement. These safety violations can be dramatic, e.g. leading to what is called a *double spending* in a *blockchain*.

Accountability, pioneered by [HKD07], is a potent property in mitigating safety violations. In the context of distributed protocols, accountability enables correct processes to conclusively detect culprits. However, in non-synchronous systems, one cannot distinguish a process whose message is delayed from a silent process and [HKD07] only provide *completeness*, i.e. every Byzantine process is eventually suspected forever by a at least one correct process.

In this paper, we consider another property requiring that safety violation implies that every correct process eventually obtains undeniable proofs of misbehaviour, which has been considered by the community only recently for some particular decision tasks like consensus [CGG$^+$] or lattice agreement [dSKRT]. We generalize theses results, by proposing a generic compiler, called τ_{scr}, that transforms any distributed protocol into an accountable version. The transformation is strongly inspired by the well-studied simulation [AW04] of crash failures on top of Byzantine ones.

2 Generic Accountability Transformation

In this section, we present our generic accountability transformation τ_{scr} that maps any non-synchronous t_0-resilient distributed protocol into its accountable counterpart.

First, we give some preliminaries on the model (section 2.1). Next, we provide an intuition behind τ_{scr} (section 2.2). Then, we overview τ_{scr} (section 2.3). Finally, we argue that τ_{scr} indeed produces an accountable counterpart of a non-synchronous distributed protocol with a quadratic overhead for the communication and message complexities by an $O(n^2)$ factore (see theorem 1).

2.1 Preliminaries

Distributed protocol We consider a set Ψ of $|\Psi| = n$ asynchronous processes that communicate by exchanging messages through a non-synchronous (either fully asynchronous or partially synchronous [DLS88]) reliable point-to-point network. Each process $p \in \Psi$ is assigned a *protocol* Π_p to follow that describe the way to send messages and to move from one state to another in accordance with messages delivery. We assume an *idealized PKI*, where each message sent by Π_p is properly authenticated. A message content m properly signed by process p is noted $\langle m \rangle_{\sigma_p}$. The adversary is assumed to be bounded, i.e., signatures of processes that follow their protocol *cannot* be forged. Processes can forward messages to other processes, they can include messages in other messages they send, and we assume that an included or forwarded message can still be authenticated. Each message m has a unique $sender(m) \in \Psi$ and a unique receiver $receiver(m) \in \Psi$. A tuple $\Pi = (\Pi_p, \Pi_q, ..., \Pi_z)$, where $\Psi = \{p, q, ..., z\}$, is a *distributed protocol*. An event is either the (local) send of some messages or the (local) reception of some messages or terminal inputs (e.g. timer expiration [DLS88]). An *execution* is a well-formed sequence of events, i.e. every received message was previously sent. Similarly, a *behavior* is a well-formed sequence of events that occur on the same process $p \in \Psi$. Given an execution α, $\alpha|_p$ denotes the sequence of events in α associated with a process $p \in \Psi$ (i.e., the behavior of p given α). A behavior β_p is *valid* according to Π_p if and only if it conforms to the assigned protocol Π_p. A process p is *correct* in an execution α according to Π if and only if $\alpha|_p$ is valid according to Π_p. Otherwise, p is *faulty* in α according to Π. A protocol can potentially ensure a conjunction of some safety properties (nothing bad occur) and a conjunction of some liveness properties (something good eventually happen) with t_0-resiliency, i.e. as long as the number of faulty processes is $t \leq t_0$.

Accountability A set M of (different) messages properly signed by a process p is a *proof of culpability* against p according to distributed protocol Π if and only if no execution α of Π exists such that p is correct and sends every message $m \in M$ in α.

Let Π be a distributed protocol that ensures safety properties P_S and liveness properties P_L with t_0-resiliency. A distributed protocol $\bar{\Pi}$ is an *accountable counterpart* of Π with factor $f \in [1, t_0]$ if there exists a homomorphic transformation $(\bar{\Pi}, \Pi, \mu_e)$ † with $\mu_e : execs(\bar{\Pi}) \rightarrow execs(\Pi)$ that satisfies (1) *Solution Preservation*, i.e. for every infinite execution $\bar{\alpha} \in execs(\bar{\Pi})$ with less than f Byzantine processes, $\mu_e(\bar{\alpha})$ ensures both P_S and P_L and (2) *Accountability*, i.e. if safety P_S is violated by $\mu_e(\bar{\alpha})$ for some $\bar{\alpha} \in execs(\bar{\Pi})$, then every correct process eventually obtains $t_0 + 1$ proofs of culpability against as many Byzantine processes according to Π.

2.2 Intuition

Consider a distributed system Ψ with $|\Psi| = n$ processes that execute a distributed protocol Π. Imagine an (unrealistic) oracle θ that belongs to the system and obtains the following abilities: All communication between processes goes through θ that forwards a message m sent by a process p only if θ observed a correct behavior from p, i.e. p followed its prescribed local protocol in accordance with the messages received by p, observed in a non-ambiguous order by θ. Upon the observation of a commission fault from p, θ ignores p forever. Such abilities allow θ to see at any point in time, an execution that is *benign*, i.e., an execution in which all processes are either correct or have crashed. Furthermore, every message m relayed by θ has a "fully-correct" causal past.

The main idea behind our τ_{scr} transformation is to simulate concepts performed by the θ oracle. We explain how that is achieved in the following subsection.

2.3 Overview

Each process is a hierarchical composition of its four layers (see figure 1):

— The state-machine layer: This layer dictates the behavior of the process, i.e., it instructs which messages are sent and which internal events are produced given the received messages and observed internal events.

†. Homomorphic transformations preserve a syntactic correspondence between $\bar{\Pi}$ and Π and some intuitive properties, e.g. a correct process in execution $\bar{\alpha}$ of $\bar{\Pi}$ remains correct in execution $\mu_e(\bar{\alpha})$ of Π

— The verification layer: The responsibility of this layer is creating a benign execution of the system (i.e., it simulates the correctness verification responsibility of θ). Specifically, the verification module builds a benign execution out of all secure-delivered messages (see the secure broadcast layer below). Observe that this layer is concerned with *all* processes of the system (whereas the state-machine layer is concerned only with the "host" process). Finally, the verification layer performs a *local* computation, i.e., it fulfills its duty irrespectively of the number of faulty processes.

— The secure broadcast layer: Every message instructed to be sent by the state-machine layer is secure-broadcast. The secure-broadcast (a.k.a. multi-shot reliable-broadcast) is a well-studied primitive of communication, handling messages of the form $\langle m, s\rangle_{\sigma_q}$ where m is the content of the message, s is a sequence number, and $\langle .\rangle_{\sigma_q}$ indicates that the whole is signed by process q. It is defined with primitives scr-bcast and scr-deliver that ensures *Integrity*, i.e. if a correct process p scr-delivers a message $\langle m, s\rangle_{\sigma_q}$ with q correct, then q has scr-bcast $\langle m, s\rangle_{\sigma_q}$, *Uniformity*, i.e. if a correct process p scr-delivers a message $\langle m, s\rangle_{\sigma_q}$, then every correct process eventually secure delivers $\langle m, s\rangle_{\sigma_q}$, *Obligation*, i.e. if a correct process p scr-bcasts a message $\langle m, s\rangle_{\sigma_q}$, then every correct process eventually scr-delivers $\langle m, s\rangle_{\sigma_q}$ and *Source Order* i.e., if a correct process scr-delivers $\langle m, s\rangle_{\sigma_q}$ then p already scr-delivered a message $\langle m', s'\rangle_{\sigma_q}$ for every $s' \in [1, s[$ and did not scr-deliver a message $\langle m'', s\rangle_{\sigma_q}$ with $m'' \neq m$. If a correct process stores at any layer the conflicting messages $\langle m, s\rangle_{\sigma_q}$ and $\langle m'', s\rangle_{\sigma_q}$ with $m \neq m''$, then it broadcasts $\{\langle m, s\rangle_{\sigma_q}, \langle m'', s\rangle_{\sigma_q}\}$ to everyone as a proof of culpability against q which is then ignored forever. Obligation and Uniformity are guaranteed only if the number of faulty processes does not exceed $\lceil n/3\rceil - 1$, while Integrity and Source Order is ensured for every $t \leq n$.

— The network layer: The layer is concerned with network manipulation (i.e., the sending and receiving of messages).

We now explain how the presented layers work in harmony to implement our τ_{scr} transformation. Let us focus on a single correct process $p \in \Psi$. Every message m instructed to be sent by the state-machine of p is (1) accompanied by the entire ongoing behavior of p up to the point of sending m (i.e., accompanied by all messages received by p thus far) ‡, and (2) secure-broadcast. In this way, p "announces" to all processes what its ongoing behavior is to allow all processes to safely verify the correctness of p. The correctness verification of p by a correct process q carries in the way imposed by θ (see section 2.2):

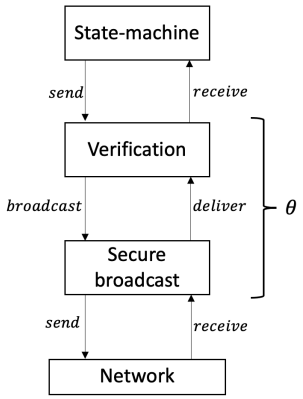

FIGURE 1: Overview of the τ_{scr} transformation

— It is checked whether the accompanied behavior is indeed correct.

— It is checked whether the accompanied behavior is a suffix of the previously verified behavior of p (this verification passes because of the order-preservation property of secure broadcast and the fact that p is correct).

— If either of the previous two verifications does not pass, process p is declared as faulty and is ignored by q in the future. In our example, p is correct, implying that it will never be declared as faulty by q.

— Process q verifies that all messages received by p in the accompanied behavior are "part" of the benign execution built by the verification module of process q §. Note that in executions with up to $\lceil n/3\rceil - 1$ Byzantine processes, since p is correct and the properties of the secure broadcast primitive hold, this condition is eventually satisfied.

— Once the last condition is fulfilled, the accompanied behavior of

‡. The transformation implementation (see [CGG+22]) introduces an optimization by piggybacking just a segment of the behavior obtained after the last message was sent, i.e., every received message is piggybacked at most *once*.

§. Since we assume that all messages are authenticated, a process cannot claim to have received a message if that is not the case.

p is included in the benign execution built by the verification module of q. Moreover, if $receiver(m) = q$, the message m is propagated to the state-machine layer of q to have q react upon the message m.

Observe that all correct processes "see" the *same* benign execution (created by their verification modules) in all non-corrupted executions (i.e., executions with up to $\lceil n/3 \rceil - 1$ Byzantine processes). More precisely, if there are less than $\lceil n/3 \rceil$ faulty processes, the verification modules of all correct processes build the same behavior of every process in the system; note that, formally speaking, observed benign executions (which are sequences of events) can *differ* in the order of events that are not causally related.

Theorem 1 (Generic Accountable Counterpart) *Let Π be a non-synchronous t_0-resilient distributed protocol. Then, $\tau_{scr}(\Pi)$ is an accountable counterpart of Π with factor $min(\lceil n/3 \rceil - 1, t_0)$. The communication complexity is multiplied by a $O(n^2)$ factor .*

PROOF SKETCH.

Since properties of secure-broadcast are ensured when $t \leq \lceil n/3 \rceil - 1$, the transformation does not tackle liveness when $t \leq min(\lceil n/3 \rceil - 1, t_0)$. Namely (1) every message sent by a correct process is eventually validated by every correct process and (2) if a message is validated by a correct process, then it will be eventually validated by every correct process, avoiding a deadlock for some correct processes during the validation procedure caused by Byzantine processes.

Recall that the verification module works correctly irrespectively of the number of faulty processes. Hence, every correct process p that adopted a behaviour β_p leading to a safety violation has observed a benign execution α^p (via its verification module) with $\beta_p = \alpha^p|_p$. If safety is violated and no more than t_0 pairs of conflicting messages are sent, it would be possible to devise an execution where t_0 faulty processes violate safety by interacting with each correct process p, *exactly* as they do in α^p. Hence, we reach a contradiction with the fact that the distributed protocol is t_0-resilient.

The quadratic overhead comes from the secure-broadcast layer. Indeed, the Uniformity property is subject to the well-known $O(n^2)$ Dolev-Reischuk bound. □

The costly overhead comes from the generality of the solution. Since we do not know which part of the generic protocol Π is subject to an attack that might violate safety, all the messages are reliable-broadcast to cover all the possibilities. For some specific protocols ([CGG+], [dSKRT]), it is possible to only reliable-broadcast some crucial messages to decrease significantly the overhead.

References

[AW04] Hagit Attiya and Jennifer L. Welch. *Distributed computing - fundamentals, simulations, and advanced topics (2. ed.).* Wiley series on parallel and distributed computing. Wiley, 2004.

[CGG+] Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. As easy as abc: Optimal (a)ccountable (b)yzantine (c)onsensus is easy! In *36th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022*.

[CGG+22] Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, Zarko Milosevic, and Adi Serendinschi. Crime and punishment in distributed byzantine decision tasks. *IACR Cryptol. ePrint Arch.*, page 121, 2022.

[DLS88] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the Association for Computing Machinery, Vol. 35, No. 2, pp.288-323*, 1988.

[dSKRT] Luciano Freitas de Souza, Petr Kuznetsov, Thibault Rieutord, and Sara Tucci Piergiovanni. Brief announcement: Accountability and reconfiguration - self-healing lattice agreement. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021*.

[HKD07] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. *SOSP'07*, 2007.