Harmful dogmas in fault tolerant distributed computing

Bernadette Charron-Bost [5] and André Schiper [6]

Abstract

Consensus is a central problem in fault tolerant distributed computing. A vast number of (positive and negative) results for consensus in various system models have been established. In this paper we isolate three features that all these system models share, and we show that inappropriate modelling choices have led to overcomplicate the approaches to studying the consensus problem, thus yielding too restrictive solutions for real systems.

It is hard to question these modelling choices, as they have gained the status of dogmas. Nevertheless, we propose a simpler and more natural approach that allows us to get rid of these dogmas, and to handle all types of benign fault, be it static or dynamic, permanent or transient, in a unified framework.

1 Introduction

Replication is the most natural technique for achieving high availability, i.e., fault tolerance in distributed systems. The technique has been studied for almost 30 years, both from a practical and a theoretical perspective. During these years, several systems have been built (e.g., the pioneering Isis system [4]) and a large number of theoretical results have been published around the paradigm of replication, namely around the consensus problem. While the goal of “system” work has been the validation of ideas through the construction of prototypes, the goal of “theoretical” work has been to provide precise definitions of problems and system models, and to identify the models in which problems are solvable. Landmark examples of theoretical work is the negative FLP result [9] (stating that consensus is not solvable deterministically in an asynchronous system with reliable links, if one process can be faulty) and positive results about the partially synchronous model [8, 13] and the asynchronous model augmented with some oracles such as random oracles [2] or failure detectors [6]. These different goals (system goals *vs.* theory goals) sometimes became antagonistic, with practical work questioning the relevance of theoretical contributions.

In this context, we claim that inappropriate modelling choices have led our community to overcomplicate the approaches to solving consensus, thus yielding too restrictive solutions for real systems. Moreover, it has become hard to question these modelling choices, as they have gained the status of dogmas: nevertheless, we suggest a much simpler approach for studying agreement

[5]Ecole polytechnique, 91128 Palaiseau, France. charron@stix.polytechnique.fr.

[6]EPFL, 1015 Lausanne, Switzerland. Andre.Schiper@epfl.ch. Research funded by the Swiss National Science Foundation under grant number 200021-111701.

problems that enables us to design general solutions and should appear more natural from a practical perspective.

The rest of the paper is structured as follows. Sections 2, 3 and 4 discuss what we claim to be the basic dogmas of fault-tolerant distributed computing. In Section 5, we briefly describe a new computation model and present its main features. Section 6 concludes the paper.

2 Distinguishing synchrony from faults: a first dogma

The FLP negative result naturally led to the definition of system models weaker than the synchronous model, in which consensus is solvable. In this context, it has become commonly accepted to define system models in terms of two parameters (cf. [12]):

- *Degree of synchrony* (what synchrony assumptions for processes and links).
- *Fault model* (what fault assumptions for processes and links).

For example, the computing model in [9] assumes that processes and links are asynchronous (degree of synchrony), links are reliable (fault model), and one process may crash (fault model). Hence degree of synchrony and fault model appear to be two independent characteristics of system models. This way of defining system types is taken for granted, but nevertheless has major drawbacks.

Firstly, consider the following three basic assumptions for a link:

1. transmission delay of a message is bounded;
2. transmission delay of a message is finite;
3. transmission delay of a message may be infinite.

Cases 1 and 2 are covered by synchrony assumptions whereas case 3 is covered by the fault model (message loss). Distinguishing synchrony assumptions from fault assumptions leads us to break the natural continuum between cases 1, 2 and 3, and tends to overcomplicate the system model space.

We can best illustrate this as follows: consider some computation in which process q is waiting for message m sent by p. If p is "too" slow, or if p has crashed, q cannot receive m since it should time-out before receiving m not to risk being blocked. Slowness of p is related to the synchrony model; crash of p is related to the fault model. Two different assumptions, same consequence for q. Similarly, consider the example from the point of view of the link from p to q. If the link behaves asynchronously "too much", or if it is lossy, q cannot receive m. The first assumption is related to the synchrony degree, the second to the fault model. Again, we have two different assumptions with the same consequence for q. This artefact directly results from the separation between synchrony model and fault model.

Secondly, the two parameter classification of system models led our community to concentrate on conditions for solving consensus with *reliable* links, while largely ignoring conditions under which consensus is solvable with *unreliable* links.[7] This is an indirect consequence of the way the FLP result has been stated. Indeed, the FLP paper shows the impossibility of solving consensus

[7]Except, for example, [8] and [13], which assume that links may be lossy but only for a finite period.

in an asynchronous system when one process may crash. As crash failure is (already) the most benign type of process failure in the classical failure classification [11], researchers have sought to circumvent the FLP impossibility result by increasing synchrony, rather than investigating other fault models. In particular, links are supposed to be reliable since message losses are usually handled by the *omission* fault model, a more severe type of fault than crashes. This leads to the wrong message that synchrony (synchronous processes and synchronous links) is sufficient to achieve agreement: as shown in [17], synchrony does not help for solving consensus in the context of link failures.

Note that the idea of encapsulating synchrony degree and fault model in the same module already appears in the *Round-by-Round Failure Detector* (for short *RRDF*) model [10]. Unfortunately, the idea is not followed through to the end in the RRFD model since the notion of fault model is underhandedly reintroduced *via* the notion of *faulty component*. Indeed, the communication medium is implicitly assumed to be reliable (no anomalous delay, no loss) and when process q receives no message from p, the latter process is systematically blamed for the transmission failure (p is late or has crashed). This point is related to the second dogma of fault-tolerant distributed computing, which we discuss in the next section.

3 Distinguishing process failures from link failures: a second dogma

The second dogma can be summarized as follows: in case of the non reception of a message, *put the responsibility on some "culprit"* (link or process). This has several drawbacks as we explain now.

First it appears that most of the time, the real causes of transmission failures, namely sender failure, receiver failure, or link failure, are actually unknown: if q does not receive a message supposed to be sent by p, it is generally impossible to know whether this is because of p (send-omission), because of q (receive-omission), or because of the (lossy) link from p to q. Failure transmissions are often ascribed to some components in a totally arbitrary manner that may not correspond to reality.

Second, as soon as a process p is blamed for a failure, p is declared *faulty*, and from here on, is allowed by any consensus specification — uniform or not — to have deviant behavior (this is discussed in more details in Section 4).

Finally, stigmatizing some components in a systematic way may lead to undesirable conclusions: for example, in the send-omission failure model, the *entire* system is considered faulty even if only *one* message from each process is lost.

There is no evidence that knowing the component responsible for the failure actually helps in the analysis of fault-tolerant systems. Even more, as we show in [7], the notion of faulty component tends to unnecessarily overloads system analysis with non-operational details. In other words, it is sufficient that the model just notifies transmission failures (effects) without specifying faulty components (causes).

Because of this second dogma, the classical models of fault tolerant distributed systems only handle faults that are *static both in space and time*, i.e., faults by an unknown but static set of (so-called faulty) processes (static faults in space) that are considered to be faulty for the whole computation (static faults in time). This explains why very few models solve consensus in the presence of *dynamic* and *transient* faults — such as message loss (possibly on all links) or crash-recovery of processes (possibly of all processes) — or when they do solve consensus, solutions are very intricate and their analysis overcomplicated. Getting rid of the second dogma gives us hope of handling any type of benign failure, be it static or dynamic, permanent or transient, in a unified framework.

In a largely ignored paper, Santoro and Widmayer [17] introduce the *Transmission Faults Model* that locates failures without specifying their cause. A transmission failure can represent as well link failure as process failure. We believe that this is the right approach. Unfortunately, the Transmission Faults Model is designed only for synchronous systems. This *de facto* reintroduces synchrony degree and fault model as two separated system parameters, i.e., the model is still based on the first dogma.

4 Definition of consensus: a third dogma

As explained above, consensus has been mainly considered in the context of static and permanent faults. With such fault models, consensus is defined by three conditions, including the following one:

- *Termination: Every correct process eventually decides.*

Termination requires processes to eventually decide, but restricts this requirement to *correct* processes (a correct process is a process that is never blamed for any transmission failure). The role of "correct" is to exempt faulty processes from deciding.

With omission failures, this termination condition allows a process blamed for just one omission to make no decision, even if it does not crash. Therefore, *termination* turns out to be a too weak condition for such a fault model. The same problem arises for any type of transient (benign) failure, e.g., the quite realistic crash-recovery model (with stable storage). For the latter fault model, Aguilera *et al.* [1] introduce the notion of *good* process — which is a process that crashes a finite number of times and recovers after the last crash — and strengthen termination as follows:

- *Termination-cr: Every good process eventually decides.*

However, it seems rather disturbing to consider different consensus specifications according to the fault model.

The same problem arises even if we do not modify the specification. Indeed with termination, depending on whether message losses are interpreted as send omission failures, receive omission failures, or as link failures, processes are exempted from making a decision, or must eventually decide. This shows that the same syntactic condition (termination) may actually correspond to different semantical requirements, according to the way we ascribe failures to components.

To design specifications whose significance do not depend on the fault model, we have to remove any reference to the notion of faulty (or correct) process. This leads us to specify termination as follows:

- *Uniform Termination: Every process eventually decides.*

Such a condition is much simpler, but it is generally considered unacceptable, with the argument that we cannot require from a crashed process or from a process that continuously crashes and recovers to decide. We shall come back to this issue in the next section: basically, we show that this is actually an artefact resulting from the confusion between an algorithm devised for some computational model and its physical implementation in some specific system.

5 An iconoclastic new model: the "Heard-Of" model

In [7] we propose a new computational model, called *Heard-Of* (*HO* for short) that is free of the three dogmas we have stated above, and so allows us to avoid the problems that they induce. We give here only a brief overview of the model, and refer to [7] for a complete presentation. Basically, in the HO model, (1) synchrony degree and fault model are encapsulated in the same abstract structure, and (2) the notion of faulty component (process or link) has totally disappeared. The HO model merely *notifies transmission failures without specifying by whom nor why* such failures occur. As a result, the conditions for solving agreement problems (1) do not to refer anymore to synchrony assumptions or to failure assumptions (dogma 1), and (2) handle process failures and link failures in the same way (dogma 2). The liveness condition for consensus is expressed using the *uniform termination* property of Section 4, which applies to any process, be it correct or faulty (dogma 3). The HO model is inspired by (1) the asynchronous round model defined by Dwork, Lynch and Stockmeyer [8], extended by Gafni [10], and by (2) the work of Santoro and Widmayer [17].

In the HO model, computation consists of *asynchronous* communication-closed rounds (a message sent but not received in round r is lost). Consider a set Π of processes. At each round, any process first sends a message to all (send phase), then receives a subset of the messages sent at this round (receive phase), and finally does some local computation (transition phase). We denote by $HO(p, r)$ the set of processes that p *hears of* at round r, i.e., the processes (including itself) from which p receives a message at round r.

An *HO model* is defined by a predicate — over the collections of sets $(HO(p, r))_{p \in \Pi, r>0}$ — that holds for all computations. For example, an HO model could be defined by the predicate:

$$\exists r_0 > 0,\ \forall p, q \in \Pi^2 \ :\ HO(p, r_0) = HO(q, r_0),$$

which ensures the existence of some round r_0 in which all processes hear of the same set of processes.

In this way, the model just describes transmission faults at each round without specifying the causes (omission or slowness), and without assigning the responsibility of these faults to some components (process, channel). These are the key features that allow us to handle any benign fault, be it static or dynamic, permanent or transient, in a unified framework.

As shown in [7], the HO formalism enables us to express well-known consensus algorithms (e.g., the *Rotating Coordinator* algorithm [6] or the *Paxos* algorithm [13]) in a quite concise and elegant way, and so to extract the algorithmic schemes on which they are based. This not only gives some new insights into these consensus algorithms, but also allows us to design new ones that are quite interesting in practice since they are correct under very realistic conditions. The consensus Algorithm 1 given below best exemplifies the latter claim. In Algorithm 1 the send phase corresponds to line 5, and the receive phase takes place between line 5 and line 6. When the receive phase terminates, the predicate on the HO's is guaranteed to hold. The local computation phase starts at line 7. In [7], we show that safety properties – namely integrity and agreement – are never violated, and that uniform termination is guaranteed by the following predicate:

$$\exists r_0 > 0,\ \exists \Pi_0, |\Pi_0| > 2n/3 : \ (\forall p \ : \ HO(p, r_0) = \Pi_0) \ \wedge \ (\forall p,\ \exists r_p > r_0 \ : \ |HO(p, r_p)| > 2n/3\,) \quad (1)$$

The predicate ensures (i) the existence of some "uniform" round (in the sense that all the heard-of sets are equal) with sufficiently large heard-of sets, and (ii) for each process p, the existence of some round $r_p > r_0$ in which $|HO(p, r_p)| > 2n/3$ holds. Note that this predicate allows rounds without any constraint on the HO's, e.g., rounds in which no messages are received.

Algorithm 1 A simple consensus algorithm in the HO model (n is the number of processes).

```
 1: Initialization:
 2:     $x_p := v_p$                                   { $v_p$ is the initial value of $p$ }

 3: Round $r$:
 4:     Send Phase:
 5:         send $\langle x_p \rangle$ to all processes

 6:     Transition Phase:
 7:         if $|HO(p,r)| > 2n/3$ then
 8:             if the values received, except at most $[\frac{n-1}{3}]$, are equal to $\overline{x}$ then
 9:                 $x_p := \overline{x}$
10:             else
11:                 $x_p :=$ smallest $x$ received
12:             if more than $2n/3$ values received are equal to $\overline{x}$ then
13:                 DECIDE($\overline{x}$)
```

The HO model has several features that distinguish it from existing models, and make it more realistic and better from a practical perspective. In particular the HO model can adapt to transient and dynamic faults. For example, liveness in Algorithm 1 just requires the existence of a "good" sequence of (at most) $n + 1$ rounds, possibly non consecutive: one uniform round r_0 in which every process hears of the same subset of processes, and for each process p, a subsequent round r_p at which p hears more than $2n/3$ processes. In the other rounds, any transient faults from any component, i.e., dynamic fault, can occur.[8] Note that the communication predicate (1) above is much finer than the usual termination conditions given in the literature, which all stipulate that some condition must eventually hold *forever* (see [8], or consensus with the failure detector $\diamond\mathcal{S}$ [6], or Paxos with reliable channels and the leader oracle Ω [5]).

Remains the issue of uniform termination which, as explained in Section 4, does not exempt any process from making a decision. Such a strong liveness requirement may seem unreasonable in two respects. First, it may make Consensus needlessly unsolvable in the sense that the resulting

[8]Specifically, this means that the *send* of line 5 can be implemented using the unreliable *IP-multicast* primitive, an option that is problematic when using failure detectors, which require reliable links.

Consensus specification might be unsolvable under some communication predicate $\mathcal{P}$ whereas the classical Consensus problem is solvable under $\mathcal{P}$. In [7], we show that this objection does not hold. Secondly, one may wonder whether an algorithm in which *all* processes decide can be implemented in real systems with crash failures. The answer is positive as we now explain. The fundamental point here is to distinguish an algorithm – a *logical* entity – from the *physical* system on which the algorithm is running. Of course, a process that has crashed takes no step, and so can make no decision. However, such a process is then mute, and so is no more heard by any other process. Consequently, what actually happens to crashed processes has no impact on the rest of the computation. Thus there is no problem when running an HO Consensus algorithm on a physical system where processes may crash: *the capability of making a decision provided by the HO algorithm is just not implemented by the processes that have crashed.* In other words, the implementation of an HO algorithm solving uniform termination in a physical system with crash failures only solves termination.

Hence we can safely remove any reference to faulty components in the Consensus specification. This yields a uniform specification in the sense that its semantics does not depend anymore on the fault model. With such a clean specification, a process is no more unreasonably exempted from making a decision just because it was once accused of a fault during the computation – maybe incorrectly. As a matter of fact, this specification of the Consensus problem already appears in several fundamental papers dealing with benign faults [15, 3, 13].

Related Work: At first sight, the HO model may seem close to the RRFD model introduced by Gafni [10]. However, these two models only share the idea of capturing synchrony degree and failure model with the same abstraction – see our first dogma – but basically differ with regard to the second and third dogmas. Indeed, the RRFD modules only suspect processes – and more specifically, only the senders – never links. On the contrary, a central idea of the HO model is to remove the notion of "culprit" (the component responsible of the fault). Ignoring culprits allows us to get rid of the typical system model assumptions such as "channels are reliable", "processes crash and do not recover" (crash-stop model), "processes crash but may later recover" (crash-recovery model), etc., and shows that solving consensus does not require to overload the system analysis with such details. In this regard, the HO model is inspired by Santoro and Widmayer [17]. Unfortunately, the idea of ignoring culprits is not completely followed through to the end in [17] since the authors assume at each round only one posssible source for all transmission faults, i.e., one process that is *responsible* for originating the transmission faults of a round. Moreover, they still consider synchrony, an assumption that does not appear in the HO model.

It should also be noted that the unification provided by the HO model must be seen from the perspective of constructing solutions to consensus that span the whole class of benign faults. This differs from other work on system models with the goal of providing unified proof of impossibility of consensus protocols, or of deriving proof of bounds on consensus, e.g., [14, 16].

6 Discussion

In the context of solving agreement problems, system models have been defined in a way that led our community to consider irrelevant details. This has obscured a technically difficult domain, making it appear unnecessarily harder than needed. The new HO model avoids the pitfalls that we have highlighted. For example, it unifies all benign faults in a quite natural way. It also makes the handling of the crash-recovery model — and more generally dynamic and transient faults — much simpler, avoiding the complexity that appears for example in [1].

The HO model also leads to a more natural expression of conditions for liveness. Previous approaches, e.g., the partially synchronous model [8] or the failure detector approach [6] require conditions to hold eventually *forever*. This is non intuitive from a pragmatic point of view. In contrast, the conditions for liveness in the HO model do not have this this problem, and appear very intuitive. This may contribute to demystify the consensus problem.

An issue not addressed here is how to ensure an HO predicate. Ongoing work has shown that the liveness condition for Algorithm 1 is easy to ensure, assuming a system that alternates between good and bad periods, a very realistic assumption: During a bad period, any benign failure can occur; during a good period, at least $2n/3$ processes can communicate timely.

To summarize, we believe that fault tolerant distributed computing can become much simpler than what inappropriate modelling choices have allowed it to be in the past.

References

[1] M.K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Computing*, 13(2):99–125, 2000.

[2] M. Ben-Or. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. In *proc. 2nd annual ACM Symposium on Principles of Distributed Computing*, pages 27–30, 1983.

[3] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *Journal of Algorithms*, 11(3):420–440, September 1990.

[4] K. Birman. The Process Group Approach to Reliable Distributed Computing. *Comm. ACM*, 36(12):37–53, December 1993.

[5] R. Boichat, P. Dutta, S. Frolund, and R. Guerraoui. Reconstructing Paxos. *ACM SIGACT News*, 34(1):47–67, 2003.

[6] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2):225–267, 1996.

[7] B. Charron-Bost and A. Schiper. The "Heard-Of" model: Unifying all benign faults. Technical Report TR, EPFL, June 2006.

[8] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of ACM*, 35(2):288–323, April 1988.

[9] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32:374–382, April 1985.

[10] Eli Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony. In *Proc of the 17th ACM Symp. Principles of Distributed Computing (PODC)*, pages 143–152, Puerto Vallarta, Mexico, June-July 1998.

[11] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. In Sape Mullender, editor, *Distributed Systems*, pages 97–145. ACM Press, 1993.

[12] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. Technical Report 94-1425, Department of Computer Science, Cornell University, May 1994.

[13] L. Lamport. The Part-Time Parliament. *ACM Trans. on Computer Systems*, 16(2):133–169, May 1998.

[14] R. Lubitch and S. Moran. Closed schedulers: A novel technique for analyzing asynchronous protocols. *Distributed Computing*, 8(4), 1995.

[15] S. Moran and Y. Wolfstahl. Extended impossibility results for asynchronous complete networks. *Information Processing Letters*, 26(3):145–151, 1987.

[16] Y. Moses and S. Rajsbaum. A layered analysis of consensus. *SIAM J. Comput.*, 31(4):989–1021, 2002.

[17] N. Santoro and P. Widmayer. Time is not a healer. In *Proceedings of the 6th Symposium on Theor. Aspects of Computer Science*, pages 304–313, Paderborn, Germany, 1989.