Model Checking of Fault-Tolerant Distributed Algorithms: from Classics towards Contemporary

Igor Konnov
University of Lorraine, CNRS, Inria, LORIA
F-54000 Nancy, France
Email: igor.konnov@inria.fr

Stephan Merz
University of Lorraine, CNRS, Inria, LORIA
F-54000 Nancy, France
Email: stephan.merz@loria.fr

***Abstract*—Fault-tolerant distributed algorithms—such as agreement, reliable broadcast, and consensus—lie at the heart of distributed systems. Although these algorithms are tiny in comparison to the rest of the system code, they are hard to design and verify. In this short research statement, we discuss the Byzantine model checker, which was developed for automatic verification of asynchronous fault-tolerant distributed algorithms. Further, we discuss the challenges that are posed by contemporary protocols for Blockchain consensus.**

1. Introduction

Fault-tolerant distributed algorithms are difficult to design right. Not surprisingly, they are even harder to verify. In [8], we noticed that there was a significant gap even between the textbook fault-tolerant distributed algorithms and the capabilities of the actual verification and bug finding tools, e.g., model checkers. The main challenges for model checking are as follows:

i. **Lack of formal specifications**: algorithms usually come in pseudo-code, and their formalization is tricky;
ii. **Parameterization**: one has to prove an algorithm being correct for all possible system sizes, e.g., independently of how many processes are in the distributed system;
iii. **Faults**: Since faults perturb the distributed system, they significantly increase the degree of non-determinism in the system and the global state space.

In Section 2, we give a short overview of the techniques that address the agenda of [8]. In Section 3, we discuss the new challenges posed by new distributed protocols such as Blockchain consensus.

2. Classics: Byzantine Model Checker

The Byzantine model checker (ByMC [2]) verifies fault-tolerant algorithms that work under the assumptions of asynchronous reliable communication: (a) a fraction of processes may fail—crash or exhibit Byzantine behavior, (b) the correct processes make infinitely many steps, and (c) a message sent by a correct process is eventually received by every correct process. The examples of such algorithms are: reliable broadcast [15], consensus in one communication step [14], atomic commitment with failure detectors [6], [13]. Importantly, these algorithms are parameterized with: the number of processes n, the upper bound on the number of faulty processes t, the number of actual faults $f \leq t$. The algorithms are resilient to faults, when the parameters satisfy the resilience condition, typically, $n > 3t$ or $n > 2t$.

In ByMC, each correct process is modelled as a threshold automaton. For example, the threshold automaton in Figure 1 models the algorithm for Asynchronous Byzantine Agreement [4]. The effect of sending a message to all processes is expressed by an increment of a shared variable, which stores the number of messages sent by the correct processes, such as x_{++} and y_{++} in our example. Thus, when $y \geq t + 1$ holds true, a correct process could have received $t + 1$ messages from the correct processes. (To enforce that every correct process eventually receives at least y messages, we write fairness constraints in linear temporal logic). Byzantine processes may add—but do not have to—up to f messages, and thus the above expression becomes $y \geq t+1-f$. The distributed system is thus modelled as a composition of threshold automata. Depending on the type of faults, we run either $n - f$ instances (Byzantine faults), or n instances (crashes). Details on our modeling decisions and threshold automata can be found in [7], [9].

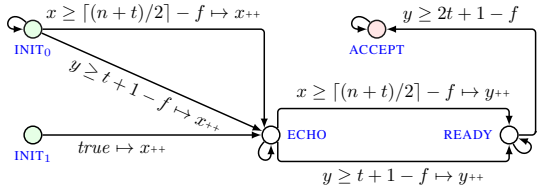

Figure 1. A threshold automaton for Byzantine agreement [4]

ByMC verifies, whether safety and liveness properties hold for the systems of threshold automata, for all the combinations of parameters that satisfy the resilience condition, e.g., $n > 3t$. This is done by encoding the bounded model checking problem with an SMT solver. Although

This research supported in part by the Vienna Science and Technology Fund (WWTF) through project APALACHE (ICT15-103).

bounded model checking is incomplete in general, ByMC relies on the short counterexample property for threshold automata [7], which states that it is sufficient to explore the executions of certain shapes within the precomputed bounds.

Importantly, ByMC provides the user with a counterexample, if there is an instance of the system that violates one of the properties. A counterexample is a sequence of steps by threshold automata for fixed parameters (e.g., for $n = 4$, $t = 1$, and $f = 1$).

Recently, the verification technique of ByMC was applied in a synthesis loop [11], in order to automatically find the thresholds, such as $t + 1 - f$ and $2t + 1 - f$ in Figure 1.

3. Contemporary: Challenges and Approaches

The approach of ByMC has been successful, due to good understanding of the computation model of reliable communication, which is a standard textbook material. The situation is different with the new protocols such as Blockchain consensus. In this note, we follow the layered view of Blockchain by Abraham & Malkhi [3].

Needless to say, when one considers Blockchain protocols instead of the textbook algorithms, the problems emphasized in [8] do not disappear. Most importantly, one needs a formal specification of the protocol just to start the verification efforts. Pseudo-code alone does not help when the underlying computational model is not well-understood. On top of that, Blockchain adds more challenges:

- **Computational and resource thresholds**. The textbook algorithms assume that the parameters n, t, and f are fixed for each run and use thresholds such as $t + 1$. Blockchain protocols use other kinds of thresholds such as the amount of computational power or the amount of resources.
- **Hashing.** Related to the thresholds, hash functions play a central role in Blockchain.
- **Multiple transactions**. In contrast to reliable broadcast [15], one cannot reason about one Blockchain transaction in isolation, but reasons about the longest chains.
- **Synchrony assumptions.** As the Byzantine processes may counterbalance their low computational power by incurring long delays, Blockchain assumes limits on the asynchrony in the system.

In view of all these challenges combined, one would probably need years of research to invent new parameterized model checking techniques similar to [7]. We believe that a more pragmatic approach would be to let the users specify the emerging protocols in a language that does not restrict the computational model at all. As a concrete example, we are assuming that fault-tolerant protocols are specified in TLA$^+$ [10], which recently received attention of distributed system designers [12].

Having specified a protocol in TLA$^+$, the users have two choices: write mechanical proofs with the TLA$^+$ proof system [5], or use a model checker such as TLC [16]. While TLC does not require much effort from the user, it is challenging for TLC to scale to complex protocols. We are currently developing the new APALACHE model checker [1], which is using bounded model checking techniques and SMT solvers as reasoning back-ends.

4. Conclusions

We hope that this note will raise awareness among the research community on the need for formal specification of the emerging protocols. Such specifications are of ultimate importance for development of new verification tools.

References

[1] "APALACHE TLA$^+$ model checker (development version)," 2017–2018, accessed: Apr, 2018. [Online]. Available: https://github.com/konnov/apalache/tree/unstable

[2] "ByMC: Byzantine model checker," 2013–2018, accessed: Apr, 2018. [Online]. Available: http://forsyte.tuwien.ac.at/software/bymc/

[3] I. Abraham, D. Malkhi *et al.*, "The blockchain consensus layer and BFT," *Bulletin of EATCS*, vol. 3, no. 123, 2017.

[4] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *J. ACM*, vol. 32, no. 4, pp. 824–840, 1985.

[5] D. Cousineau, D. Doligez, L. Lamport, S. Merz, D. Ricketts, and H. Vanzetto, "TLA$^+$ proofs," in *18th Intl. Symp. Formal Methods (FM 2012)*, ser. LNCS, D. Giannakopoulou and D. Méry, Eds., vol. 7436. Paris, France: Springer, 2012, pp. 147–154.

[6] R. Guerraoui, "Non-blocking atomic commit in asynchronous distributed systems with failure detectors," *Distributed Computing*, vol. 15, no. 1, pp. 17–25, 2002.

[7] I. Konnov, M. Lazić, H. Veith, and J. Widder, "A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms," in *POPL*, 2017, pp. 719–734.

[8] I. Konnov, H. Veith, and J. Widder, "Who is afraid of Model Checking Distributed Algorithms?" 2012, contribution to: CAV Workshop $(EC)^2$. http://forsyte.at/download/ec2-konnov.pdf.

[9] ——, "What you always wanted to know about model checking of fault-tolerant distributed algorithms," in *PSI 2015, Revised Selected Papers*, ser. LNCS, vol. 9609. Springer, 2016, pp. 6–21.

[10] L. Lamport, *Specifying systems: the* TLA$^+$ *language and tools for hardware and software engineers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[11] M. Lazic, I. Konnov, J. Widder, and R. Bloem, "Synthesis of distributed algorithms with parameterized threshold guards," in *OPODIS*, ser. LIPIcs, vol. 95, 2017, pp. 32:1–32:20.

[12] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How Amazon web services uses formal methods," *Comm. ACM*, vol. 58, no. 4, pp. 66–73, 2015.

[13] M. Raynal, "A case study of agreement problems in distributed systems: Non-blocking atomic commitment," in *HASE*, 1997, pp. 209–214.

[14] Y. J. Song and R. van Renesse, "Bosco: One-step Byzantine asynchronous consensus," in *DISC*, ser. LNCS, vol. 5218, 2008, pp. 438–450.

[15] T. Srikanth and S. Toueg, "Simulating authenticated broadcasts to derive simple fault-tolerant algorithms," *Dist. Comp.*, vol. 2, pp. 80–94, 1987.

[16] Y. Yu, P. Manolios, and L. Lamport, "Model checking TLA$^+$ specifications," in *Correct Hardware Design and Verification Methods*. Springer, 1999, pp. 54–66.