

Abstractions for Maintainable Byzantine Fault Tolerance

Marko Vukolić*

IBM Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
mvu@zurich.ibm.com

1 Introduction

Theoretical foundations of Byzantine fault-tolerance (BFT) were established nearly three decades ago [9, 8]. Nevertheless, it took nearly 20 years for BFT to gain the attention of systems researchers [3]. However, and perhaps surprisingly, even after a decade of intensive systems research, it seems that BFT is not much closer to a practical instantiation than when it was at its infancy.

Some immediate potential 'rationales' behind this might actually be misleading. Whereas the BFT failure model might appear way too general, this is not evident in the context of the today's web-based computing and the advent of cloud computing. Similarly, there is an increasing number of proofs that the cost of BFT solutions and their performance degradation relative to their crash-tolerant counterparts are actually acceptable (e.g., [6, 1]). So the roots of the problem seem to stem from elsewhere.

In this talk, we look for them in the context of the BFT state-machine replication, a fundamental BFT application. We identify and address two often neglected aspects of BFT: *maintenance* and *abstractions*. Like any software, if indeed adopted in a practical setting, a BFT protocol must be maintainable; in particular, easily and provably upgradable. In contrast, the current tendency is to implement a new BFT protocol either from scratch or by modifying an existing solution in a completely non-modular manner. Such non-modular modifications with upgrades in mind might even chain (e.g., [3]-[7]-[11]), typically ending up with more than 20.000 lines of sophisticated C code. This reflects the difficulties that BFT researchers face when they wish to upgrade their protocols and the great need for reusable abstractions that will facilitate such upgrades. In this talk we propose and advocate one such abstraction: *ABSTRACT*: ABortable STate mACHine Replication.

2 The abstraction

Abstract's specification looks like that of a replicated state machine; nevertheless, *Abstract* may sometimes *abort* a client request. If it does, it returns a digest of which requests had committed so far which can be further used by a client proxy to pursue the computation using another *Abstract*

*This is joint work with Rachid Guerraoui (EPFL) and Vivian Quéma (CNRS)

instance, transparently from the perspective of the actual client. From there on, further requests from that client go directly to this instance (until this, in turn, possibly aborts). The condition under which *Abstract* is allowed to abort models a progress semantics and is a generic parameter. We build a BFT protocol as a dynamic composition of instances of our abstraction. Each instance, intended for specific operating conditions, is developed, proved and tested independently. Required progress semantics for normal (common case) operating conditions, will cast *Abstract* outside the scope of the FLP [5] impossibility, allowing for very efficient *Abstract* implementations.

From a different perspective, *Abstract* formalizes and 'standardizes' (by defining the appropriate API) the concept of *views* and *view changes*, used in virtually every BFT protocol to date. Just like changing views, changing the *Abstract* instance allows system reconfiguration. However, while a typical BFT protocol merely reiterates a single protocol across different views (where only replica leader is changed), *Abstract* allows simple and flexible *switching* among different protocols, which can be individually tailored and tuned for particular operating conditions. In a sense, *Abstract* switching provides the unprecedented ability to change the entire BFT protocol on a 'view-change'.

Abstract facilitates upgrades of BFT state machine replication algorithms by allowing black-box reuse of the code, simplifying testing and even allowing model checking (to some extent) of BFT protocols. As an example of our approach, we were able to design a new BFT protocol using *Abstract* that outperforms existing ones in key performance metrics (latency and throughput) under common-case system conditions, without sacrificing the reliability of the protocol in the worst-case conditions (e.g., as in [4]). The size of the new code, developed exclusively to address common-case operating conditions, is an order of magnitude smaller than that of a modern BFT protocol.

While *Abstract* has been initially designed with strongly consistent replication semantics in mind, the work is ongoing to port *Abstract* to weak-consistency [10, 11], another ineluctable item in the BFT research agenda [2].

The above speaks much in favor of applicability of *Abstract* in bringing BFT protocols closer to the real world. However, one point is particularly important with this goal in mind: *Abstract* is an abstraction and abstractions are nice. We use abstractions all the time when teaching students, the future practitioners, about the fundamentals of computing. Arguably, BFT world is very complex and badly needs the larger number of adequate abstractions to facilitate its enlargement beyond the research boundaries.

References

- [1] Lorenzo Alvisi, Allen Clement, Mike Dahlin, Manos Kapritsos, Sang Min Lee, Taylor Riche, and Yang Wang. BFT you can believe in. In *SOSP '09: Proceedings of twenty-second ACM SIGOPS symposium on Operating systems principles*. To appear, 2009.
- [2] Ken Birman, Gregory Chockler, and Robbert van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, 2009.
- [3] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

- [4] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 153–168, Berkeley, CA, USA, 2009. USENIX Association.
- [5] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
- [6] James Hendricks, Gregory R. Ganger, and Michael K. Reiter. Low-overhead Byzantine fault-tolerant storage. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 73–86, New York, NY, USA, 2007. ACM.
- [7] Ramakrishna Kotla, Allen Clement, Edmund Wong, Lorenzo Alvisi, and Mike Dahlin. Zyzzyva: speculative byzantine fault tolerance. *Commun. ACM*, 51(11):86–95, 2008.
- [8] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
- [9] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.
- [10] Atul Singh, Pedro Fonseca, Petr Kuznetsov, Rodrigo Rodrigues, and Petros Maniatis. Defining weakly consistent Byzantine fault-tolerant services. In *LADIS '08: Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, pages 1–5, New York, NY, USA, 2008. ACM.
- [11] Atul Singh, Pedro Fonseca, Petr Kuznetsov, Rodrigo Rodrigues, and Petros Maniatis. Zeno: eventually consistent Byzantine-fault tolerance. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 169–184, Berkeley, CA, USA, 2009. USENIX Association.