ABCperf: Performance Evaluation of Fault Tolerant State Machine Replication Made Simple: Demo Abstract

Tilo Spannagel
tilo.spannagel9@kit.edu
KASTEL, KIT
Karlsruhe, Germany

Marc Leinweber
marc.leinweber@kit.edu
KASTEL, KIT
Karlsruhe, Germany

Adriano Castro
adriano.castro9@kit.edu
KASTEL, KIT
Karlsruhe, Germany

Hannes Hartenstein
hannes.hartenstein@kit.edu
KASTEL, KIT
Karlsruhe, Germany

Abstract

We demonstrate a framework that simplifies the performance evaluation of fault tolerant State Machine Replication in the permissioned model. ABCperf offers a message passing abstraction with eventual delivery on top of which interchangeable Atomic Broadcast algorithms and decentralized applications can be independently implemented. Varying network quality (i.e., latency, packet loss) and attacker behavior (i.e., omission faults) can be directly configured and are emulated by the ABCperf core. The framework allows the real-time manipulation of configuration options and visualizes performance indicators and statistics in real time.

CCS Concepts: • **Computer systems organization → Reliability**; **Availability**; • **Theory of computation → Distributed algorithms**.

Keywords: Emulation, Fault injection, Performance analysis

ACM Reference Format:
Tilo Spannagel, Marc Leinweber, Adriano Castro, and Hannes Hartenstein. 2023. ABCperf: Performance Evaluation of Fault Tolerant State Machine Replication Made Simple: Demo Abstract. In *24th International Middleware Conference Demos, Posters and Doctoral Symposium (Middleware Demos, Posters and Doctoral Symposium '23), December 11–15, 2023, Bologna, Italy.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3626564.3629101

1 Introduction

State Machine Replication (SMR) is suited to implement decentralized applications where all stakeholders contribute equally to the decentralized computing system. The Atomic Broadcast (ABC) algorithm used to synchronize the state machine replicas defines the maximum number of faulty peers f depending on n, the number of stakeholders. Even if the resulting f seems to be reasonably high, every fault, including simple faults like omission, may have an impact on the overall performance as they typically trigger additional coordination and communication. Recent work has shown that the performance of an ABC algorithm may degrade severely when getting close to the tolerable number of crashed replicas [1]. Thus, load that is easily handled under ideal conditions may overload the system when facing simple faults (i.e., omission) that are not necessarily caused by an attacker. The mere knowledge of f is not sufficient to design a reliable SMR-based decentralized application. Rather, precise knowledge of the algorithm behavior under various conditions is necessary and the ability to evaluate differing system designs for a use case is needed.

Narwhal [1] is a current work suggesting a state-of-the-art algorithm that evaluates confirmation latency and throughput of different ABC algorithms under crash faults. Others, e.g., [6], evaluate under datacenter conditions with no faults occurring. If code is available and can be run, e.g., [1], it is work-intensive and error-prone to extend the experiment setup to another algorithm for comparison, implying the need for a generic performance evaluation framework. Existing evaluation frameworks [2–4] are designed for the comparison of complete blockchain ecosystems and require a full-fledged replica software. In particular, it is not sufficient to only supply the ABC algorithm as an experiment input. BFT-Bench [4] and Blockbench [2] are not designed for extensibility and are only capable of emulating crash faults. Diablo [3] is extendable to new Blockchain systems but not capable of network and fault emulation. The existing frameworks output various statistical values but do not offer live observation or raw data analysis.

We demonstrate ABCperf, a performance evaluation framework and middleware. ABCperf provides everything necessary to allow the implementation of ABC algorithms at an abstraction level similar to pseudocode. In order to evaluate algorithms under conditions close to real-world setups, ABCperf supports the implementation of interchangeable replicated state machines and the corresponding load emulation. It is capable of emulating network latency, packet loss, and omission faults following configurable probability distributions. We decided to emulate omission faults as they are a powerful fault type that can be generically generated. In most protocols using cryptographic primitives, Byzantine behavior is reduced to omission nonetheless (correct replicas drop invalid messages). To observe the impact of changing environmental conditions and attacker behavior, ABCperf features a real-time interaction module that visualizes the system under investigation and allows the adjustment of configuration parameters. As our research focuses on ABC algorithms that make use of Trusted Execution Environments, we created a MinBFT [7] and a preliminary TEE-Rider implementation [5] that we use to demonstrate the viability of ABCperf. The ABCperf and MinBFT implementations are open source[1].

2 ABCperf

ABCperf is written in Rust, chosen for its performance and type safety, and consists of an orchestrator, a core, and three interface type definitions. The orchestrator is a dedicated ABCperf service that reads the configuration, assists in establishing the peer-to-peer network, schedules experiments, generates client requests, and collects measurements. In an experiment, each replica uses ABCperf's peer-to-peer middleware (i.e., the core) providing two

[1] https://github.com/abcperf

Middleware Demos, Posters and Doctoral Symposium '23, December 11–15, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0429-1/23/12…$15.00
https://doi.org/10.1145/3626564.3629101

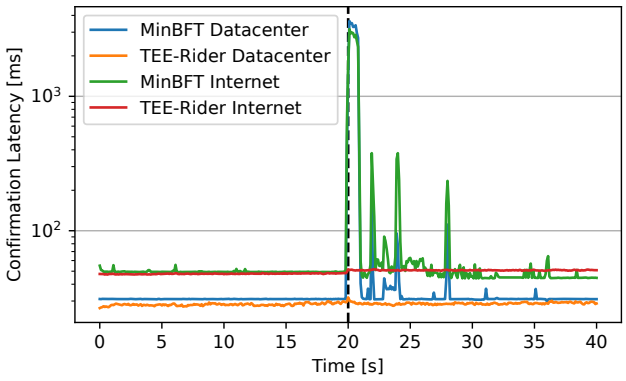

Figure 1. Confirmation latency of MinBFT and TEE-Rider (10 replicas, 250 requests/s). After 20s (vertical line), replica 0 drops all future messages. For datacenter conditions, all replicas are connected via the same switch. Internet-like conditions are a normally distributed latency ($\mu = 4.94$ms, $\sigma = 2.52$ms) and a packet loss of 0.128%.

features: First, it offers a message passing interface with eventual delivery, abstracting the complete communication stack and allowing to send arbitrary messages to replicas by addressing them with simple integer IDs. Second, it is capable of emulating both attacker and network behavior transparently. Attacker behavior, i.e., omission faults, is emulated by dropping messages from or to replicas marked as faulty following an adjustable probability distribution. Latency and packet loss are emulated using the capabilities of netem[2]. ABCperf can be deployed to a cluster (multiple replicas per physical server are possible), letting it scale with the hardware. However, our experiments revealed a bottleneck in the replica-to-replica communication code that is currently investigated.

The users of ABCperf are required to implement three interface types that are used by core (`AtomicBroadcast` and `Application`) and orchestrator (`ClientEmulator`). `AtomicBroadcast` is an interface for ABC algorithms and makes simple replacement of ABC algorithms possible whilst encouraging a clean implementation at an abstraction level similar to pseudocode. Users of ABCperf are only required to implement algorithm state, message handling and induced state transition, as well as message generation. In particular, the transport protocol, serialization, peer discovery, or parallelization do not have to be considered. The code implementing the `Application` interface implements the decentralized end-user application and is responsible to handle generated client requests. ABCperf is capable of running full-fledged decentralized apps. Nonetheless, they are typically simplified to facilitate the emulation of client requests (e.g., allowing simpler patterns of consecutive requests). The orchestrator generates client requests using an implementation of the `ClientEmulator` interface corresponding to a decentralized application. The request types, sequences, and timings can be chosen freely, allowing complex client behavior.

The orchestrator measures each client request and collects utilization (i.e., CPU, memory, and network usage) as well as algorithm measurements from the replicas. The raw data of the measurements is stored in a database to allow flexible and complex analysis. Additionally, the orchestrator offers a REST API that allows live manipulation of configuration parameters and that supplies the aforementioned metrics. The demonstrator is a web application that connects to the orchestrator's REST API. The metrics graphed in real-time are confirmation latency and throughput over time, both being the primary performance indicators of a decentralized application. Furthermore, the CPU, memory, and network usage of each replica are illustrated and, if applicable, the current leader is highlighted. The web application allows the adjustment of network and omission fault emulation as well as the choice of faulty replicas. An example for the confirmation latency of MinBFT [7] and TEE-Rider [5] under non-ideal conditions is depicted in Fig. 1. The parameters for the emulation of Internet conditions are derived from RIPE Atlas[3] for Central Europe. After 20s of experiment time, replica 0 drops every future incoming and outgoing message. In case of MinBFT, this leads to a view change, making replica 1 the new leader, and to a corresponding increase in latency caused by a temporary interruption of request handling. Under Internet conditions, the impact of the view change overhead is higher and lasts longer in comparison to datacenter conditions, where all replicas are connected via the same switch, minimizing network latency while maximizing network throughput. TEE-Rider, being a leaderless algorithm, tolerates the omission of replica 0 considerably better and only has a very small but ongoing latency penalty caused by the randomized leader election choosing replica 0. Under datacenter conditions, TEE-Rider outperforms MinBFT.

In conclusion, ABCperf simplifies the evaluation of fault tolerant state machine replication and visualizes the effects of non-ideal conditions. We are currently finalizing implementations of state-of-the-art partially synchronous and asynchronous ABC algorithms and will compare them in a range of conditions that vary by load, network connectivity, and fault patterns.

Acknowledgments

This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF).

References

[1] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman. 2022. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*. ACM, 34–50. https://doi.org/10.1145/3492321.3519594

[2] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. ACM, 1085–1100. https://doi.org/10.1145/3035918.3064033

[3] V. Gramoli, R. Guerraoui, A. Lebedev, C. Natoli, and G. Voron. 2023. Diablo: A Benchmark Suite for Blockchains. In *Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys 2023, Rome, Italy, May 8-12, 2023*. ACM, 540–556. https://doi.org/10.1145/3552326.3567482

[4] D. Gupta. 2016. *Towards Performance and Dependability Benchmarking of Distributed Fault Tolerance Protocols*. Ph. D. Dissertation. Grenoble Alpes University, France. https://tel.archives-ouvertes.fr/tel-01376741

[5] M. Leinweber and H. Hartenstein. 2023. Brief Announcement: Let It TEE: Asynchronous Byzantine Atomic Broadcast with n ≥ 2f+1. In *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy (LIPIcs, Vol. 281)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 43:1–43:7. https://doi.org/10.4230/LIPIcs.DISC.2023.43

[6] J. Liu, W. Li, G. O. Karame, and N. Asokan. 2019. Scalable Byzantine Consensus via Hardware-Assisted Secret Sharing. *IEEE Trans. Computers* 68, 1 (2019), 139–151. https://doi.org/10.1109/TC.2018.2860009

[7] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Veríssimo. 2013. Efficient Byzantine Fault-Tolerance. *IEEE Trans. Computers* 62, 1 (2013), 16–30. https://doi.org/10.1109/TC.2011.221

[2] https://man7.org/linux/man-pages/man8/tc-netem.8.html

[3] https://atlas.ripe.net/