Challenging Anti-fragile Blockchain Applications

Miguel González[1], Walter Rudametkin[1], Martin Monperrus[1], and Romain Rouvoy [1,2]
[1]Univ. Lille / Inria
[2]IUF

***Abstract*—Failures in production are a *de facto* rule for distributed software systems. In particular, modern distributed systems are composed of heterogeneous building blocks contributed by third parties and guaranteeing the end-to-end resilience is becoming a major challenge. Even though each of these software components can embed fault tolerance or dependability protocols, it remains difficult to assess their effectiveness upon the occurrences of unexpected failures.**

As part of this work, we propose a new generation of fault injection framework that can be deployed in production to challenge Blockchain-based distributed systems. This paper therefore reports on the state of the art in this area and potential opportunities for novel contributions towards building anti-fragile distributed systems on the Blockchain.

I. Introduction

As failures are unavoidable in computer systems, engineers have to embrace it. In such complex systems, many things can go wrong: disks failures, DIMMs failures, networks are unreliable, engineers make mistakes, and therefore software have bugs. Distributed systems are difficult to program and to debug because there is a certain unpredictability, about their executions and outcomes. Fault tolerance and resilience are important characteristics that this kind of systems must have. Message passing, the essence of distributed systems, can lead to data inconsistencies due to asynchronous communication channels, nondeterminism in the ordering and timing of message delivery.

The Blockchain [1] is a distributed database technology invented for the digital currency Bitcoin, yet offers to benefit many industries other than the financial industry. Some of the use cases of a Blockchain are: tracking digital use, supply chain, voting, banks, content distribution and smart contracts. The Blockchain is a paradigm shift for organizing processes with less friction and more efficiency, and at higher scale than current paradigms. More specifically, *permissioned Blockchains* aim to provide higher scalability and performance (more transactions per second and less latency), by removing the *Proof-of-Work* (PoW) consensus and using *Byzantine Fault Tolerant* (BFT) protocols instead, along with the addition of an identity management node.

The decentralization is what gives the Bitcoin network great resilience and thus it becomes anti-fragile [2]. One could argue that removing or altering some of Bitcoin's core properties like moving away from a fully decentralized architecture would make it a more fragile protocol, but there is actually a trade-off to be made for applications wishing to use Blockchain technologies in different contexts. This trade-off directly affects scalability, security, redundancy, speed, etc., but it also affects, more indirectly, many of the anti-fragile properties that are popular with the Bitcoin protocol itself. This research activity intends to focus on developing a framework that aids developers to debug, verify and reason about distributed systems that leverage the emerging permissioned Blockchain technologies. In particular, We want to extract the key features from highly resilient technologies, in order to build a general framework that is useful for testing Blockchain-based distributed systems in production.

In the remainder of this paper, we comment on the state of the art in the context of fault tolerance, resilience and verification through fault injection and a problem statement my thesis aims to address that expands on the concept of the Blockchain, followed by a short conclusion.

II. State of the Art

Most of the biggest IT companies, like Google, Amazon, Microsoft, Netflix have had failures leading to major service outage [3]. These systems are supposedly designed to be highly available, generally developed using good software engineering practices, and intensely tested. This begs the question of why these systems still fail and what can be done to improve their resilience. For example, configuration errors and incorrect exception handling have been found to affect distributed systems.[4]

Distributed systems should adapt in the presence of failures, this concept sometimes is defined interchangeably as fault tolerance or resilience. [5] Fault tolerant systems only bring the system to an execution state [6]. Resilience is the persistence of service delivery when facing changes [5]. The concept of anti-fragility means a system should grow stronger from each successive stressor, disturbance, and failure. Anti-fragile software extracts the intrinsic value of errors and learns from them [7].

McCaffrey [8] lists the different ways to verify distributed systems. Among them are formal verification methods and lineage driven fault injection. Distributed model checkers and distributed tracing are also means to verify distributed systems but are out of scope in this document. Fault injection is another way to verify distributed systems by perturbing and observing the system behavior. It is a relatively mature subject in the literature [3] and seems to be the dominant approach in the software engineering community, because it can quickly identify superficial bugs without much effort. Other literature identifies heuristic fault injection and byzantine fault injection and instrumentation techniques.

Formal methods. One solution is to use formal verification methods like TLA+ and Coq have been shown to have verified the AWS infrastructure and S3 storage [9]. However, event if formal methods are powerful, they may not scale well and require extensive expertise in the modeling language.

Random fault injection. Netflix has been pushing random failure injection in *production* [10] with their *SimianArmy* and *Chaos Monkey* frameworks. Tests are run in production because it's never possible to fully reproduce all aspects of the system in a testbed. Because the search space of

failure scenarios is too large, random fault injection might produces good enough results, but wastes resources in testing "uninteresting" faults: complex combinations of failures are not explored by this model.

Heuristic fault injection. Gunawi et al. [11] presented FATE, a fault-injection framework targeting the recovery logic in cloud applications, systematically exploring failure scenarios with multiple-failure injection. FATE uses a combination of brute force and heuristic search (ranking) to explore failure combinations and scenarios, but like the random methodology, they do little to cover the space of possible executions.

Instrumentation. Ju et al. [12] created a fault-injection framework that targets service communication between OPENSTACK (a cloud computing platform) services, leveraging execution graphs built from logs to monitor and observe the processing of external requests. Instrumentation is done between OPENSTACK services. The problem with this approach is that the execution graphs are not complete due to logging issues and the heavy use of instrumentation required.

Byzantine fault injection. Martins et al. [13] developed HERMES, a fault injection framework that provides a baseline to test and debug BFT protocols. This work tackles Byzantine failures (when fault nodes can pretend to be correct nodes by generating arbitrary data) in distributed systems, which have been excluded in the rest of the literature, the issue with them is that they only target a set of predefined protocols and not full blown systems.

Lineage-driven fault injection. Alvaro et al. [14] introduced *Lineage-driven Fault Injection* (LDFI). LDFI uses data lineage to reason backwards (from effects to causes), but only works with DEDALUS (a logic programming language) programs. This algorithm was later implemented at Netflix and the results showed improvement of an order of magnitude against random fault injection [15], replacing lineage provided by DEDALUS with a distributed tracing system, like DAPPER [16]. The downside is the lineage collection itself, tracing systems can only give so much information and LDFI depends on the quality of the input data.

III. PROBLEM STATEMENT

Currently, banks and other institutions are gyrating towards Blockchain technologies, but being a recent distributed technology, the implementations are untested and will likely fail at some point. Also, depending on the choices made, the specific protocols chosen may have unforeseen weaknesses (e.g., lack of redundancy, slow throughput, DoS). Despite their importance, Blockchain technologies do not have any fault injection frameworks, nor have they been formally verified. Available fault-injection solutions do not cover Byzantine failures, the ones that do require too much instrumentation and only target BFT protocols not Blockchain solutions. Most Blockchain technologies rely on a BFT protocol to achieve consensus.

This work will focus on creating a framework that leverages some of the best approaches of current injection models described in the literature, and use the framework to perturb and verify permissioned Blockchain technologies with Byzantine failures, like Hyperledger[1], in production. This endeavor presents a few challenges, for example building a general solution to inject Byzantine failures into Blockchains despite their differences in language or architecture. Because we will focus on perturbing applications that are considered highly resilient, we expect the results to be useful in the construction of anti-fragile distributed systems in general.

IV. CONCLUSION

We reported a brief summary on the state of the art of the efforts to build and increase resilience in distributed systems, we identified a problem in the current dependability literature and propose creating a new fault injection framework for systems built using Blockchain-based technologies.

REFERENCES

[1] M. Swan, *Blockchain: Blueprint for a new economy*. "O'Reilly Media, Inc.", 2015.

[2] R. Pérez-Marco, "Blockchain tme and Heisenberg Uncertainty Principle." working paper or preprint, Nov. 2016.

[3] H. S. Gunawi, T. Do, J. M. Hellerstein, I. Stoica, D. Borthakur, and J. Robbins, "Failure as a service (faas): A cloud service for large-scale, online failure drills," tech. rep., EECS Department, University of California, Berkeley, Jul 2011.

[4] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. U. Jain, and M. Stumm, "Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, USENIX Association, 2014.

[5] J.-c. Laprie, "From dependability to resilience," in *In 38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, 2008.

[6] A. D. Keromytis, "Characterizing self-healing software systems," in *In Proceedings of the 4th International Conference on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS*, 2007.

[7] M. Monperrus, "Principles of antifragile software," *CoRR*, vol. abs/1404.3056, 2014.

[8] C. McCaffrey, "The verification of a distributed system," *Queue*, vol. 13, Dec. 2015.

[9] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How amazon web services uses formal methods," *Commun. ACM*, vol. 58, Mar. 2015.

[10] A. Tseitlin, "The antifragile organization," *Commun. ACM*, vol. 56, Aug. 2013.

[11] H. S. Gunawi, T. Do, P. Joshi, P. Alvaro, J. Yun, J.-s. Oh, J. M. Hellerstein, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, K. Sen, and D. Borthakur, "Fate and destini: A framework for cloud recovery testing," tech. rep., EECS Department, University of California, Berkeley, Sep 2010.

[12] X. Ju, L. Soares, K. G. Shin, K. D. Ryu, and D. Da Silva, "On fault resilience of openstack," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, ACM, 2013.

[13] R. Martins, R. Gandhi, P. Narasimhan, S. Pertet, A. Casimiro, D. Kreutz, and P. Veríssimo, *Experiences with Fault-Injection in a Byzantine Fault-Tolerant Protocol*. Springer Berlin Heidelberg, 2013.

[14] P. Alvaro, J. Rosen, and J. M. Hellerstein, "Lineage-driven Fault Injection," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, ACM, 2015.

[15] P. Alvaro, K. Andrus, C. Sanden, C. Rosenthal, A. Basiri, and L. Hochstein, "Automating Failure Testing Research at Internet Scale," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, ACM, 2016.

[16] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," tech. rep., Google, Inc., 2010.

[1] https://www.hyperledger.org