

Indicus: Unchaining Byzantine Databases

Your N. Here
Your Institution

Second Name
Second Institution

Abstract

Your abstract text goes here. Just a few facts. Whet our appetites. Not more than 200 words, if possible, and preferably closer to 150.

1 Introduction

There is an increasing tension between the desire to share data online, and the security concerns it entails. This paper asks the question: how can we enable *mutually distrustful parties* to consistently and reliably share data, while minimizing centralization?

The ability to share data online offers exciting opportunities. In banking, systems like SWIFT enable financial institutions to quickly and accurately receive information such as money transfer instructions; and in manufacturing, online data sharing can improve accountability and auditing amongst the globally distributed supply chain.

Increased data sharing, however, raises questions of how to *decentralize trust*. Banking institutions must currently place their trust in the centralized SWIFT's network to issue payment orders. Sometimes there is even no identifiable source of trust. Consider the supply chain for the latest iPhone: it spans three continents, and hundreds of different contractors []; neither Apple nor these contractors trust each other, yet all must be willing to agree and share information about the construction of the same product.

Recognizing this challenge by both the research and industry communities, much effort has focused on enabling shared computation between mutually distrustful parties, in the context of byzantine fault tolerance (BFT), and blockchains. Systems proposed in the literature of BFT[][] provide the abstraction of a totally ordered log; the log is agreed upon by the n participants in the system, of which at most f can misbehave. Each participant executes operations that may touch one to multiple objects in the log. In the blockchain world, Bitcoin and Ethereum have become popular distributed computing platforms providing the same log abstraction and aiming

for decentralizing trust. Furthermore, Microsoft Azure has launched projects[] that leverage these blockchain platforms and extend the digital transformation beyond the companies' four walls in a supply chain.

This paper argues that there exists a fundamental mismatch between the implementation of a totally ordered log and the reality of much large-scale distributed processing. Many large-scale distributed systems consist primarily of unordered and unrelated operations. For example, a product supply chain consists of many concurrent steps that do not require ordering. Imposing an ordering on non-conflicting operations is not only often unnecessary, but costly: participants in the shared computation must vote to order operations, store the full state of the system, and replay the full log for auditing.

While there exists work on mitigating this scalability bottleneck through sharding [], the latent total order requirement introduces unnecessary coordination overhead, as coordination is performed twice, at the level of individual shards, and across shards. Callinicos [] and Omniledger [], for instance, runs a full BFT protocol for every operation. This is especially problematic when workloads are geo-replicated [], or when, as in BFT, the replication factor is high. Further, these systems support transactions under the assumption that their read and write operations are known a priori, which limits the set of applications that they can support.

As another research trend of mitigating the scalability bottleneck, EPaxos[], TAPIR[] and CURP[] only consider the ordering between potentially conflicting operations, instead of commutative operations. However, these systems assume the crash-failure model and are non-trivial to be extended to the Byzantine model, so that they cannot directly solve the problem of data sharing among mutually distrustful parties.

Existing research, in essence, is either attempting to build concurrency control and sharding functionalities over BFT replication, or integrating these functionalities into a crash-failure replication protocol. In this paper, we will show how to build these desiring functionality inside a BFT replication protocol. Specifically, our goal is to *provide the illusion of a centralized shared log, rather than the non-scalable reality of*

a totally ordered log.

Sorry.

2 Background

[?]

3 Indicus

3.1 Architecture

3.2 Model and Definitions

3.3 Life cycle

3.4 Consistent Logging

3.5 Concurrency Control

- Optimization: retries - Dependency resolution

3.6 Writeback and Multi-shard 2pc

- Optimization: Single shard logging

3.7 Failures

4 Implementation and Evaluation

5 Limitations

6 Related Work/Comparison

7 Conclusion

Acknowledgments

The USENIX latex style is old and very tired, which is why there's no \acks command for you to use when acknowledging.

Availability

USENIX program committees give extra points to submissions that are backed by artifacts that are publicly available. If you made your code or data available, it's worth mentioning this fact in a dedicated section.

References

- [1] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. *ACM SIGOPS Operating Systems Review*, 41(6):45–58, 2007.